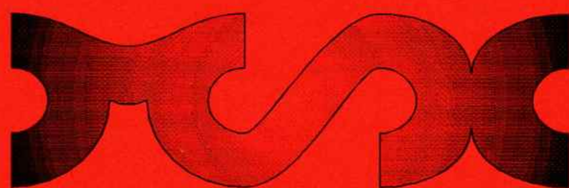
TURBO



PASCAL

HANDLEIDING

Inhoudsopgave

1. BASIS TAALELEMENTEN

1.1 Basis symbolen

De wezenlijke vocabulaire van Turbo Pascal bestaat uit symbolen, die in letters, cijfers en speciale symbolen ingedeeld kunnen worden:

Letters: A tot Z, a tot z en _ (onderstreping)
Cijfers: 0 1 2 3 4 5 6 7 8 9
Speciale symbolen: + - * / = ^ < > [ ] ( ) { } . , : ; ' # $

Tussen hoofd- en kleine letters wordt geen onderscheid gemaakt. Bepaalde operatoren en begrenzers worden uit twee speciale symbolen opgebouwd:

Toewijzingsoperator: :=
Relationele operatoren: <> <= >=
Deelbereik begrenzers: ..
Haakjes: ( ) kunnen in plaats van [ ] gebruikt worden.
Commentaar: (* *) kunnen in plaats van { } gebruikt worden.

1.2 Gereserveerde woorden

Gereserveerde woorden zijn een vastliggend onderdeel van Turbo Pascal en kunnen niet opnieuw gedefinieerd worden. Ze kunnen niet door de gebruiker als gedefinieerde namen worden gebruikt. De gereserveerde woorden zijn:

*absolute	*external	nil	*shl
and	file	not	*shr
array	forward	*overlay	*string
begin	for	of	then
case	function	or	type
const	goto	packed	to
div	*inline	procedure	until
do	if	program	var
downto	in	record	while
else	label	repeat	with
end	mod	set	*xor

Tabel 1 - 1

In de gehele handleiding zijn de gereserveerde woorden **vet** gedrukt. De sterretjes geven gereserveerde woorden aan die in de Pascal standaard niet gedefinieerd zijn.

1. BASIS TAALELEMENTEN

1.1 Basis symbolen

De wezenlijke vocabulaire van Turbo Pascal bestaat uit symbolen, die in letters, cijfers en speciale symbolen ingedeeld kunnen worden:

Letters: A tot Z, a tot z en _ (onderstreping)

Cijfers: 0 1 2 3 4 5 6 7 8 9

Speciale symbolen: + - * / = ^ < > [ ] ( ) { } . , : ; ' # $

Tussen hoofd- en kleine letters wordt geen onderscheid gemaakt. Bepaalde operatoren en begrenzers worden uit twee speciale symbolen opgebouwd:

Toewijzingsoperator: :=

Relationele operatoren: <> <= >=

Deelbereik begrenzers: ..

Haakjes: ( ) kunnen in plaats van [ ] gebruikt worden.

Commentaar: (* *) kunnen in plaats van { } gebruikt worden.

1.2 Gereserveerde woorden

Gereserveerde woorden zijn een vastliggend onderdeel van Turbo Pascal en kunnen niet opnieuw gedefinieerd worden. Ze kunnen niet door de gebruiker als gedefinieerde namen worden gebruikt. De gereserveerde woorden zijn:

*absolute	*external	nil	*shl
and	file	not	*shr
array	forward	*overlay	*string
begin	for	of	then
case	function	or	type
const	goto	packed	to
div	*inline	procedure	until

do	if	program	var
downto	in	record	while
else	label	repeat	with
end	mod	set	*xor

Tabel 1 - 1
In de gehele handleiding zijn de gereserveerde woorden vet gedrukt. De sterretjes geven
gereserveerde woorden aan die in de Pascal standaard niet gedefinieerd zijn.

1.3 Standaard namen

Turbo Pascal heeft de volgende standaard namen voor voorgedefinieerde typen,
constanten, variabelen, procedures en functies. Al deze namen kunnen opnieuw
gedefinieerd worden, maar dat betekend, dat hun mogelijkheden worden beperkt. Zulke
veranderingen kunnen gemakkelijk tot verwarring leiden. De volgende standaard namen
kunnen daarom het best in hun oorspronkelijke definitie worden gelaten:

Addr	Delay	Length	Release
ArcTan	Delete	Ln	Rename
Assign	EOF	Lo	Reset
Aux	EOLN	LowVideo	Rewrite
AuxInPtr	Erase	Lst	Round
AuxOutPtr	Execute	LstOutPtr	Seek
BlockRead	Exit	Mark	Sin
BlockWrite	Exp	MaxInt	SizeOf
Boolean	False	Mem	SeekEof
BufLen	FilePos	MemAvail	SeekEoln
Byte	FileSize	Move	Sqr
Chain	FillChar	New	Sqrt
Char	Flush	NormVideo	Str
Chr	Frac	Odd	Succ
Close	GetMem	Ord	Swap

ClrEol	GotoXY	Output	Text
ClrScr	Halt	Pi	Trm
Con	HeapPtr	Port	True
ConInPtr	Hi	Pos	Trunc
ConOutPtr	IOResult	Pred	UpCase
Concat	Input	Ptr	Usr
ConstPtr	InsLine	Random	UsrInPtr
Copy	Insert	Randomize	UsrOutPtr
Cos	Int	Read	Val
CrtExit	Integer	ReadLn	Write
CrtInit	Kbd	Real	WriteLn
DelLine	KeyPressed		

Tabel 1 - 2

In de gehele handleiding worden de standaard namen, zoals ook alle andere namen (zie hoofdstuk 3.1 "Namen") met hoofd- en kleine letters geschreven. In de tekst worden ze cursief gedrukt.

1.4 Begrenzers

Taalelementen moeten minstens door ÈÈn van de volgende begrenzers gescheiden worden: spatie, tab, nieuwe regel of commentaar.

1.5 Programma regels

Programma regels mogen maximaal 127 tekens lang zijn, meer tekens worden door de compiler genegeerd. In eerdere TURBO versies liet de editor ook maar 127 tekens op ÈÈn regel toe. Nu er gebruik gemaakt moet worden van een externe editor zou de source langere regels kunnen bevatten. In zo'n geval worden de tekens na kolom 127 genegeerd, let u er dus op dat de regels korter blijven.

2. SCALAIRE STANDAARD TYPEN

Een data type definieert de aard van de waarden, die een variabele kan aannemen. Iedere variabele in een programma moet ÈÈn data type hebben. Hoewel data typen in Turbo Pascal zeer complex kunnen zijn, worden ze allemaal uit eenvoudige (ongestructureerde) typen opgebouwd.

Een eenvoudig type kan of door de programmeur gedefinieerd zijn (het type wordt dan gedeclareerd scalair type genoemd), of het is ÈÈn van de scalaire standaard typen:
Longint, Integer, Real, Boolean, Char of Byte. Er volgt een beschrijving van deze zes scalaire data typen.

2.1 Integer (hele getallen)

Integers zijn hele getallen. In Turbo Pascal hebben deze het bereik van -32768 tot 32767.
Hele getallen nemen twee bytes geheugen in beslag.

Overlopen van rekenkundige operaties met hele getallen wordt niet ontdekt. Let u er vooral op, dat tussenresultaten in Integer expressies binnen de bereikgrenzen voor hele getallen moeten liggen. Zo geeft bijvoorbeeld de expressie 1000*100/50 niet 2000, omdat de vermenigvuldiging tot een overloop leidt.

2.2 Byte

Het type Byte is een deelbereik van het type Integer met als grenzen 0 en 255. Bytes zijn derhalve compatibel met het type Integer, dat betekend dat wanneer een waarde van het type Byte verwacht wordt er ook een Integer gebruikt mag worden en omgekeerd. Een uitzondering hierop geldt bij de overdracht van parameters. Verder kunnen Bytes en Integers in expressies gemengd worden en Byte variabelen kunnen integer waarden toegewezen krijgen. Een variabele van het type Byte neemt ÈÈn byte geheugen in beslag.

2.3 Real (reÎle getallen)

Het bereik van reÎle getallen (data type real) is 1E-38 tot 1E+38. De mantisse heeft tot 11 significante cijfers. De opslag van een reÎel getal neemt 6 bytes geheugen in beslag.

Bij een rekenkundige operatie met reÎle getallen veroorzaakt een overloop een programmastop en de melding van een looptijdfout (runtime error). Een onderschrijding van de bereikgrens leidt tot een resultaat van nul.

Hoewel het type Real tot de scalaire data typen hoort, zou het volgende onderscheid tussen het type Real en de andere scalaire typen opgemerkt moeten worden:

1) De functies Pred en Succ mogen geen reÎle getallen als argument ontvangen.

2) Het type Real mag niet bij de indexering van arrays gebruikt worden.

3) Het type Real kan niet gebruikt worden om het basis type van een set te definiÎren.

4) Het type Real kan niet in de lus opdrachten for en case gebruikt worden.

5) Deelbereiken van het type Real zijn niet toegestaan.

2.4 Boolean (boolse waarheidswaarden)

Een boolse waarheidswaarde kan alleen de logische waarden waar of niet waar, waarvan de standaard namen True respectievelijk False zijn, aannemen. Deze zijn zo gedefinieerd dat True > False. Een boolean variabele neemt ÈÈn byte geheugen in beslag. De ordinale waarde van True is 1, en van False is 0.

2.5 Char (alfanumerieke tekens)

De (alfanumerieke) waarde van het type Char komt overeen met een teken uit de ASCII-tekenset. De tekens zijn volgens hun ASCII-waarde gerangschikt, bijvoorbeeld 'A' < 'B'. De ordinale waarden (ASCII-waarden) van de tekens loopt van 0 tot 255. Een Char neemt een byte geheugen in beslag.

3. ZELFGEDEFINIEERDE TAALELEMENTEN

3.1 Namen

Namen (Engl.: identifier) worden gebruikt, om labels, constanten, typen, variabelen, procedures en functies mee te identificeren. Een naam bestaat uit een letter of een onderstreping, gevolgd door een willekeurige combinatie van letters, cijfers en onderstrepingen. De lengte van een naam wordt alleen begrenst door het aantal tekens (127) dat op ÈÈn regel mag staan en alle letters van de naam zijn significant voor de compiler.

Voorbeelden:

TURBO

vierkant

getelde_personen

GeboorteDatum

3eMacht   mag niet omdat er een cijfer aan het begin staat.

twee Woorden   mag niet omdat er geen spatie in mag zitten.

Omdat Turbo Pascal geen verschil maakt tussen hoofd- en kleine letters, heeft het gebruik
van hoofd- en kleine letters bij bijvoorbeeld GeboorteDatum geen invloed. Het gebruik
van hoofd- en kleine letters wordt echter toch aangeraden, omdat het de leesbaarheid
verhoogd. Voor het menselijk oog is ZeerLangeNaam nu eenmaal gemakkelijker te lezen
dan ZEERLANGENAAM. De gemengde toepassing van hoofd- en kleine letters wordt
door de hele handleiding voor namen gebruikt.

3.2 Getallen

Gehele getallen

Getallen zijn constanten van het type Longint, Integer of Real. Integer constanten zijn
hele getallen, die in decimale of hexadecimale notatie worden uitgedrukt. Hexadecimale
getallen worden gekenmerkt doordat ze vooraf gegaan worden door een dollarteken:
$ABC is een hexadecimale constante. Het bereik van de decimale Longint is van
-2147483648 tot en met +2147483647 en het hexadecimale Longint bereik loopt van
$00000000 tot en met $FFFFFFFF. Het bereik van de decimale Integer is van -32768 tot
en met +32767 en het hexadecimale Integer bereik loopt van $0000 tot en met $FFFF.

Voorbeelden:

1

12345

-1

$123

$ABC

$123G   mag niet omdat G geen bestaand hexadecimaal cijfer is.

1.2345   mag niet omdat een heel getal geen punt mag bevatten.

ReÎle getallen

Het type Real heeft een bereik van 1E-38 tot en met 1E+38. Dit betekend een mantisse van 11 significante cijfers. U kunt de exponent schrijfwijze gebruiken, waarbij de letter E de exponent factor vooraf gaat en betekend '10 tot de macht'. Een Integer constante is overal geoorloofd waar een Real constante mag staan. Er mogen geen scheidingstekens binnen het getal staan.

Voorbeelden:

1234.5678

-0.012

1E6

2E-5

-1.2345678901E+12

1   toegestaan, maar dit is geen Real, maar een Integer.

3.3 Strings

Een String constante is een opeenvolging van tekens, met enkelvoudige aanhalingstekens er omheen:

'Dit is een string constante'

Een enkelvoudig aanhalingsteken kan in de String staan doordat het dubbel geschreven wordt. Strings die maar ÈÈn enkel teken bevatten gelden als het standaard type Char. Een String is met een array of char van dezelfde lengte compatibel. Alle string constanten zijn met alle String typen compatibel.

Voorbeelden:

'TURBO'

'You''ll see'

''''

';'

''

Zoals in het tweede en derde voorbeeld te zien is, wordt een enkelvoudig aanhalingsteken
in een String als twee op elkaar volgende aanhalingstekens geschreven. De vier op elkaar
volgende aanhalingstekens in het derde voorbeeld stelt een String voor met ÈÈn enkel
aanhalingsteken.

Het laatste voorbeeld - aanhalingstekens die geen tekens omsluiten betekenen een lege
String - is alleen met het String type compatibel.

3.4 Controle karakters

Turbo Pascal biedt ook de mogelijkheid om controle karakters in een String op te nemen.
Er zijn twee manieren om deze op te nemen in een String:

1. Het teken # gevolgd door een gehele constante binnen het bereik van 0 tot en met 255
komt overeen met het ASCII-karakter van dezelfde waarde.

2. Het teken ^ gevolgd door een letter komt overeen met het overeenkomstige controle
karakter, ASCII-waarde is ordinale waarde van de letter (hoofdletters) min 64.

Voorbeelden:

#10   ASCII 10 decimaal regel opvoer   = Line Feed

#$1B   ASCII 1B hex   = Escape

^G   Control-G   = Bell

^L   Control-L pagina opvoer   = Form Feed

^[Control-[= Escape

Opeenvolgende controle karakters kunnen als een String worden samengevoegd door ze
te schrijven zonder begrenzers tussen de verschillende controle karakters.

```
#13#10

#27^U#20

^G^G^G^G
```

De bovenstaande Strings bevatten respectievelijk 2, 3 en 4 karakters. Controle karakters
kunnen ook worden gecombineerd met tekst Strings:

```
'Ik wacht op invoer! '^G^G^G' Wakker worden'

#27'U'

'Dit is de volgende regel met tekst '^M^J
```

Deze drie regels bevatten respectievelijk 37, 3 en 37 karakters.

3.5 Commentaar

Commentaar mag overal in de tekst worden opgenomen waar begrenzers toegestaan zijn.
Commentaar moet worden omsloten door de tekens { } of door (* *).

Voorbeelden:

```
{dit is commentaar }

(* en dit ook *)
```

Commentaar geschreven tussen { } mag niet omsloten worden door ander commentaar
dat met { } is omsloten en voor commentaar dat omsloten is met (* *) geldt hetzelfde.
Commentaar binnen { } mag echter wel omsloten worden door (* *) en omgekeerd.
Hierdoor is het mogelijk om hele stukken van de broncode als commentaar op te nemen,

zelfs als binnen dit stuk al commentaar staat.

3.6 Compiler aanwijzingen

Een aantal mogelijkheden van de Turbo Pascal compiler worden gestuurd door compiler aanwijzingen (compiler directives). Een compiler aanwijzing wordt in de broncode opgenomen als commentaar dat op een speciale manier is geschreven. Een compiler aanwijzing mag dus overal voorkomen waar commentaar mag staan.

Een compiler aanwijzing bestaat uit een openingshaakje direct gevolgd door een dollarteken en weer direct gevolgd door een compiler aanwijzing letter of een lijst letters gescheiden door komma's. De schrijfwijze van de letter of van de lijst letters hangt af van het type aanwijzing. Een volledige omschrijving van iedere compiler aanwijzing vindt u in de appendix C.

Voorbeelden:

```
{$I-}

{$i include.fil}

{$R-, B+, V+}

(*$X-*)
```

Merk op dat er geen spaties voor of achter het dollarteken mogen staan, anders wordt het opgevat als commentaar.

4. PROGRAMMAHOOFD EN BLOK

Een Pascal programma bestaat uit een programmahoofd, gevolgd door een programmablok. Het programmablok is verder onderverdeeld in een declaratie deel, waarin alle in het programma voorkomende objecten gedefinieerd worden, en een uitvoeringsdeel, waarin alle acties gespecificeerd worden, die met deze objecten uitgevoerd moeten worden. Beide delen worden in het navolgende nauwkeurig beschreven.

4.1 Programmahoofd

Bij Turbo Pascal is het programmahoofd optioneel en heeft geen betekenis. Indien

aanwezig, geeft het aan het programma een naam en kan de parameters opsommen,
waardoor het programma met zijn omgeving communiceerd. Deze lijst bestaat uit een rij
van namen, die tussen haakjes staat en door komma's gescheiden zijn.

Voorbeelden:

```
program Cirkels;

program BoekHouding (Input, Output);

program Schrijver (Input, Printer);
```

4.2 Declaratie deel

Het declaratie deel van een blok, declareert alle namen, die in het opdrachtdeel van het
blok (en mogelijkerwijs andere blokken binnen dit blok) gebruikt worden. Het declaratie
deel is in vijf verschillende delen ingedeeld:

1) Label declaratie deel

2) Constante definitie deel

3) Type definitie deel

4) Variabelen declaratie deel

5) Procedure- en/of functie declaratie deel

Terwijl de standaard Pascal voorschrijft dat elk blok of niet, of slechts eenmaal mag
voorkomen in de bovenstaande volgorde, laat Turbo Pascal toe dat elk van de delen
willekeurig vaak en in elke volgorde in het declaratie deel mogen optreden.

4.2.1 Label declaratie deel

Elke opdracht in het programma mag van een label worden voorzien, wat het mogelijk
maakt met een goto opdracht direct naar deze opdracht toe te springen. Een label bestaat
uit een labelnaam, die gevolgd wordt door een komma. Voor het gebruik moet eerst het
label declaratie deel gedeclareerd worden. Het gereserveerde woord label staat aan het
begin van dit deel, daarna volgt een lijst met labelnamen, die met komma's gescheiden
worden waarna het geheel afgesloten wordt door een puntkomma.

Voorbeeld:

```
label 10, Fout, 999, Afbreek;
```

Terwijl standaard Pascal de labels tot getallen van vier cijfers beperkt, mag men met
Turbo Pascal zowel getallen als namen als label gebruiken.

4.2.2 Constanten definitie deel

Het constante definitie deel voert namen als synoniemen in voor de constante waarden.
Het gereserveerde woord const staat aan het begin van dit deel, daarna volgt een lijst met
constanten toewijzingen, die door puntkomma's gescheiden worden. Elke constante
toewijzing bestaat uit een naam, waarop een is-teken en de constante waarde volgt.
Constanten zijn of Strings, of getallen, die, zoals beschreven in hoofdstuk 3.2 "Getallen"
en 3.3 "Strings", gedefinieerd zijn.

Voorbeeld:

```
const

  Limiet = 255;

  CodeWoord = 'SESAM';

  CursHome = ^['V';
```

De volgende constanten zijn voor gedefinieerd in Turbo Pascal en kunnen worden
gebruikt zonder voorafgaande declaratie:

Naam

Type

Waarde

Pi

Real

3.1415926536E+00

False

Boolean

Logische waarde False,
ordinale waarde 0

True

Boolean

Logische waarde False,
ordinale waarde 0

Maxint

Integer

32767

MaxLongint

Longint

2147483647

Tabel 4 - 1

Zoals beschreven is in hoofdstuk 13. "FILE TYPEN", mogen in een constante definitie
ook getypeerde constanten worden opgenomen.

4.2.3 Type definitie deel

Een data type in Pascal mag zowel direct beschreven worden in het variabele declaratie
deel, als via een naamsverwijzing in het type definitie deel. Er zijn verschillende
standaard typen voorhanden, en de programmeur kan zijn eigen type definiÎren via het
type declaratie deel. Het gereserveerde woord type gaat vooraf aan het type definitie deel
en wordt gevolgd door ÈÈn of meer type toekenningen, welke gescheiden worden door
puntkomma's. Elke type toekenning bestaat uit een naam, gevolgd door het is-gelijk teken
(=) en een type.

Voorbeelden:

type

```
  Getal = Integer;

  Dag   = (Maan, Dins, Woens, Donder, Vrij, Zater, Zon);

  Lijst = array[1..10] of Real;
```

Meer voorbeelden van type definities vindt u in de volgende hoofdstukken.

4.2.4 Variabelen declaratie deel

Elke variabele die in het programma voorkomt moet worden gedeclareerd. De declaratie moet in de tekst plaatsvinden voordat de variabele gebruikt wordt, zodat de compiler de variabele 'kent'.

Een variabele declaratie bestaat uit het gereserveerde woord var gevolgd door ÈÈn of meer namen, gescheiden door een komma, en gevolgd door een dubbele punt en een type. Hiermee ontstaat een nieuwe variabele van het gespecificeerde type en aangeduid met zijn eigen naam.

De variabele met een naam is toegankelijk binnen het blok waarin de naam is gedeclareerd, en elk subblok binnen dit blok. Merk echter wel op dat binnen elk subblok in dit blok de naam opnieuw kan worden gebruikt voor het declareren van een andere variabele. De variabele wordt lokaal ten opzichte van het subblok genoemd (en elk subblok binnen dit subblok), en de variabele die op een hoger niveau is gedeclareerd (de globale variabele) wordt ontoegankelijk.

Voorbeeld:

```
var

  resultaat, Balans, SubTotaal : Real;

  I, J, X, Y                   : Integer;

  Aanvaard, Toegestaan         : Boolean;

  Buffer                       : array[0..127] of Byte;
```

4.2.5 Procedure en functie declaratie deel

Het procedure declaratie deel wordt gebruikt om een procedure binnen een blok of
programma (zie hoofdstuk 15.2.1) te definiÎren. Een procedure wordt aangeroepen met
een procedure opdracht (zie hoofdstuk 6.1.2) en na afloop wordt het programma hervat
met de opdracht die volgt direct op de procedure opdracht.

Een functie declaratie wordt gebruikt om een programmadeel te definiÎren dat een
waarde berekend en aflevert (zie hoofdstuk 15.3). Een functie wordt uitgevoerd als zijn
naam als onderdeel van een expressie (zie hoofdstuk 5.2) is gebruikt.

4.2.6 Opdrachtendeel

Het opdrachtendeel is het laatste deel van een blok. Hierin staan de opdrachten in
volgorde van uitvoering opgenomen. Het volledige opdrachtendeel heeft dezelfde vorm
als een samengestelde opdracht gevolgd door een punt.

Een samengestelde opdracht begint met het gereserveerde woord begin, gevolgd door
een lijst met opdrachten die gescheiden worden door een puntkomma en afgesloten
worden met het gereserveerde woord end.

5. EXPRESSIES

Expressies zijn algoritmische constructies, die regels voor de berekening van waarden
aangeven. Ze bestaan uit operanden, dat zijn variabelen, constanten en functienamen, die
door operatoren gecombineerd worden.

Dit hoofdstuk beschrijft, hoe de expressies met de standaard typen Integer, Longint, Real,
Boolean en Char gemaakt worden. Expressies, die de gedeclareerde, scalaire typen,
String- en Set typen bevatten worden respectievelijk in hoofdstuk 7.1, 8.2 en 11.2
behandeld.

5.1 Operatoren

Operatoren vallen in vijf categorieÎn, die hier op prioriteit geordend zijn:

1) Monadische min (min met maar ÈÈn operand).

2) Not operator.

3) Vermenigvuldigingsoperatoren: *, /, div, mod, and, shl en shr.

4) Opteloperatoren: +, -, or en xor.

5) Relationeleoperatoren: =, <>, <, >, <=, >= en in.

Operatoren van dezelfde prioriteit worden van links naar rechts afgehandeld. Expressies tussen haakjes worden eerst berekend, onafhankelijk van de operatoren hierna of hiervoor.

Het resulterend type van een vermenigvuldigings- of optellingsoperator is altijd gelijk aan het type met de hoogste significantie.

Type
Integer
Longint
Real
Integer
Integer
Longint
Real
Longint
Longint
Longint
Real
Real
Real
Real
Real

Tabel 5 - 1

5.1.1 Monadische min

De monadische min betekend een teken omkering van zijn operand, deze kan van het

type Integer, Longint of Real zijn (positief wordt negatief en andersom).

5.1.2 Not operator

De not operator keert de logische waarde om van zijn boolse operand:

not True = False

not False = True

Turbo Pascal laat ook toepassing van de not operator toe op operanden van het type
Integer en Longint, in dat geval wordt een bitgewijze omkering uitgevoerd.

Voorbeelden:

not 0   = -1   (Longint = $FFFFFFFF , Integer = $FFFF)

not -15   = 14

not $2345 = $DCBA

5.1.3 Vermenigvuldigingsoperatoren

Operator

Bewerking

Operand type

Resulterend type

*

Vermenigvuldiging

Real

Real

*

Vermenigvuldiging

Longint

Longint

*

Vermenigvuldiging

Integer

Integer

*

Vermenigvuldiging

Real, Longint

Real

*

Vermenigvuldiging

Real, Integer

Real

*

Vermenigvuldiging

Longint, Integer

Longint

/

Deling

Real

Real

/

Deling

Longint

Real

/

Deling

Integer

Real

/

Deling

Real, Longint

Real

/

Deling

Real, Integer

Real

/

Deling

Longint, Integer

Real

div

Integer deling

Integer

Integer

div

Integer deling

Longint

Longint

div

Integer deling

Integer, Longint

Longint

mod

Modulo

Integer

Integer

mod

Modulo

Longint

Longint

mod

Modulo

Integer, Longint

Longint

and

Rekenkundige and

Integer

Integer

and

Rekenkundige and

Longint

Longint

and

Rekenkundige and

Integer, Longint

Longint

and

Logische and

Boolean

Boolean

shl

Schuif links

Integer

Integer

shl

Schuif links

Longint

Longint

shr

Schuif rechts

Integer

Integer

shr

Schuif rechts

Longint

Longint

Tabel 5 - 2
Voorbeelden:

```
12 * 34        = 408
123 / 4        = 30.75
123 div 4      = 30
12 mod 5       = 2
True and False = False
2 and 22       = 4
2 shl 7        = 256
256 shr 7      = 2
```

5.1.4 Opteloperatoren

Operator

Bewerking

Operand type

Resulterend type

+

Optellen

Real

Real

+

Optellen

Integer

Integer

+

Optellen

Longint

Longint

+

Optellen

Real, Integer

Real

+

Optellen

Real, Longint

Real

+

Optellen

Longint, Integer

Longint

-

Aftrekken

Real

Real

-

Aftrekken

Integer

Integer

-

Aftrekken

Longint

Longint

-

Aftrekken

Real, Integer

Real

-

Aftrekken

Real, Longint

Real

-

Aftrekken

Longint, Integer

Longint

or

Rekenkundige or

Longint

Longint

or

Rekenkundige or

Integer

Integer

or

Logische or

Boolean

Boolean

xor

Rekenkundige xor

Longint

Longint

xor

Rekenkundige xor

Integer

Integer

xor

Logische xor

Boolean

Boolean

Tabel 5 - 3

Voorbeelden:

123 + 456 = 579

456 - 123.0 = 333.0

True or False = True

12 or 22 = 30

True xor False = True

12 xor 22 = 26

5.1.5 Relationele operatoren

Relationele operatoren gelden voor alle scalaire standaard typen: Integer, Real, Boolean,
Char en Byte. Operanden van het type Integer, Real en Byte kunnen gemengd worden.
Het type van het resultaat is altijd Boolean, dat betekend True of False (waar of onwaar).

= is gelijk

<> is ongelijk

> is groter dan

< is kleiner dan

>= is groter of gelijk

<= is kleiner of gelijk

Voorbeelden:

a = b is waar, als a gelijk is aan b

a <> b is waar, als a ongelijk is aan b

a > b is waar, als a groter is dan b

a < b is waar, als a kleiner is dan b

a >= b is waar, als a groter dan of gelijk is aan b

a <= b is waar, als a kleiner dan of gelijk is aan b

5.2 Functie aanroep

De functie aanroep is een functienaam, die door een parameterlijst gevolgd kan worden.
De parameterlijst kan bestaan uit constanten, variabelen en/of expressies, die door komma's gescheiden zijn en door haakjes omgeven moeten worden. Een functie wordt geactiveerd als binnen een expressie de naam van de functie voorkomt. Als die functie geen standaard functie is moet deze voor de activering gedeclareerd worden.

Voorbeelden:

Round (PlotPos)

WriteLn (Pi*(Sqr(R)))

(Max (X, Y) < 25) and (Z > Sqrt (X*Y))

Volume (Radius, Hoogte)

6.   OPDRACHTEN

Het opdrachtdeel definieert de actie die door het programma (of het subprogramma) uitgevoerd moet worden. Elke opdracht specificeert een deel van de actie. In die zin is Pascal een sequentiÎle taal: de opdrachten worden sequentieel afgehandeld, nooit tegelijk. Het opdrachtdeel is omsloten met de gereserveerde woorden begin en end, waarbinnen de opdrachten door puntkomma's gescheiden zijn. Opdrachten kunnen enkelvoudig of gestructureerd zijn.

6.1 Enkelvoudige opdrachten

Enkelvoudige opdrachten bevatten geen andere opdrachten. Enkelvoudige opdrachten zijn de toewijzings-, procedure-, goto- en de lege opdracht.

6.1.1 Toewijzingsopdracht

De meest elementaire opdracht van alle opdrachten is de toewijzingsopdracht. Deze wordt gebruikt, om aan te geven dat aan een bepaalde variabele een bepaalde waarde toegewezen moet worden. Een toewijzing bestaat uit een variabelenaam, de toewijzingsoperator := gevolgd door een expressie.

Toewijzingen zijn voor variabelen van elk type (behalve bestanden) mogelijk, zolang de variabele en de expressie van hetzelfde type zijn. Als uitzondering hierop mag bij een Real variabele, de expressie van het type Integer of Longint zijn, en bij een Longint mag de expressie van het type Integer zijn.

Voorbeelden:

```
Hoek      := Hoek * Pi;

ToegangOk := False;

Toegang   := Antwoord=Password;

BolVol    := 4*Pi*R*R;
```

6.1.2 Procedure opdracht

De procedure opdracht dient ervoor, een voordien door de gebruiker gedefinieerde of voor gedefinieerde standaard procedure te activeren. De opdracht bestaat uit een procedurenaam, al of niet gevolgd door een parameterlijst. Deze parameterlijst is een lijst van variabelen of expressies, die door komma's gescheiden en door haakjes omgeven zijn. Als bij de uitvoering van het programma de procedure opdracht bereikt wordt, wordt de controle aan de procedure overgedragen, de waarden van de eventuele parameters worden ook aan de procedure overgedragen. Als de procedure afgelopen is, gaat de uitvoering van het programma verder met de opdracht, die op de procedure aanroep volgt.

Voorbeelden:

```
Zoek (Naam, Adres);

Sorteer (Adres);
```

```
HoofdLetters (Tekst);

WerkFileBij (KlantKaart);
```

6.1.3 Goto opdracht

Een goto opdracht bestaat uit het gereserveerde woord goto, waarop een labelnaam volgt.
De goto opdracht zorgt ervoor dat de sequentiÎle verwerking van het programma
onderbroken wordt, en verder gaat op de positie die met de labelnaam is gemarkeerd. De
volgende regels moeten bij het gebruik van goto in acht genomen worden.

1) Voor gebruik moeten labels gedeclareerd worden. De declaratie gebeurd in het label
declaratie deel van het blok, waarin ze gebruikt wordt.

2) De reikwijdte van het label is het blok, waarin het gedefinieerd werd. Daarom is het niet
mogelijk, in of uit procedures en functies te springen.

6.1.4 Lege opdracht

Een lege opdracht bevat geen symbolen en heeft geen werking. Deze mag voorkomen
waar Pascal een opdracht verwacht, maar waar geen acties plaats moeten vinden.

Voorbeelden:

```
begin end.

while Antwoord<>'' do;

repeat until KeyPressed;   {wacht tot er een toets wordt ingedrukt}
```

6.2 Gestructureerde opdracht

Gestructureerde opdrachten zijn constructies die uit andere opdrachten samengesteld zijn.
Deze worden sequentieel (samengestelde opdrachten), voorwaardelijk (voorwaardelijke
opdrachten) of herhaald (herhalende opdrachten) uitgevoerd. De discussie van de with
opdracht wordt verschoven naar hoofdstuk 10.2.

6.2.1 Samengestelde opdracht

Een samengestelde opdracht wordt gebruikt als de Pascal syntax maar ÈÈn opdracht
toestaat maar er meerdere opdrachten gewenst. Het bestaat uit een willekeurig aantal

opdrachten, die met puntkomma's gescheiden zijn en door de gereserveerde woorden
begin en end ingesloten worden. De aparte opdrachten van de samengestelde opdracht
worden op de volgorde, waarin ze geschreven zijn, afgehandeld.

Voorbeeld:

```
if Kleinste > Grootste then

begin

  Tijdelijk := Kleinste;

  Kleinste  := Grootste;

  Grootste  := Tijdelijk;

end;
```

6.2.2 Voorwaardelijke opdracht

Een voorwaardelijke opdracht kiest ÈÈn van zijn deel opdrachten voor de uitvoering uit.

6.2.2.1 If opdracht

De if opdracht (beslissing) specificeert dat een opdracht alleen uitgevoerd moet worden
als aan een bepaalde voorwaarde (boolse expressie) voldaan (True) is. Als er niet aan
voldaan is (False), dan wordt of geen opdracht uitgevoerd, of wordt de opdracht die op
het gereserveerde woord else volgt uitgevoerd. Let erop, dat aan else geen puntkomma
vooraf mag gaan.

De syntactische tweeduidendheid die uit de volgende constructie ontstaat:

```
if expr1 then

  if expr2 then

    stmt1

  else

    stmt2
```

wordt vermeden, doordat het als volgt geÔnterpreteerd wordt:

```
if expr1 then

begin

  if expr2 then

    stmt1

  else

    Stmt2

end
```

Dat betekend dat het else deel altijd hoort bij de laatste if opdracht, die geen else deel heeft.

Voorbeelden:

```
if Rente>25 then

  OpMaken:=True

else

  NeemLening:=OK;

if (Invoer<0) or (Invoer>100) then

begin

  WriteLn ('Invoer bereik ligt tussen 1 en 100.');

  Write ('Opnieuw invoeren S.V.P. :');

  Read (Invoer)

end;
```

6.2.2.2 Case opdracht

De case opdracht (keuze) bestaat uit een expressie (de sorteerder) en een lijst met opdrachten, die steeds door een case label (eenvoudige expressie) van het type van de sorteerder voorafgegaan wordt. Deze geeft aan dat de opdracht waarvan het label met de actuele waarde van de sorteerder overeenkomt, uitgevoerd moet worden. Als geen enkele

case label de waarde van de sorteerder heeft, dan worden of geen of eventueel de
opdrachten die na het gereserveerde woord else volgen uitgevoerd. De else bij de case
opdracht is een uitbreiding van de standaard Pascal.

Een case label bestaat uit een willekeurig aantal van constanten of deelbereiken, die door
komma's gescheiden zijn en die gevolgd worden door een dubbele punt. Een deelbereik
wordt als twee constanten geschreven die gescheiden worden door de
deelbereikbegrenzer '..'. Het type van de constanten moet gelijk zijn aan het type van de
sorteerder. De opdracht, die op het case label volgt, wordt uitgevoerd als de waarde van
de sorteerder gelijk is aan ÈÈn van de constanten, of in een deelbereik ligt.

Geldige typen voor de sorteerder zijn alle enkelvoudige typen, dat zijn alle scalaire typen
met uitzondering van het type Real.

Voorbeelden:

```
case Operator of

  '+': Resultaat:=Antwoord+Resultaat;

  '-': Resultaat:=Antwoord-Resultaat;

  '*': Resultaat:=Antwoord*Resultaat;

  '/': Resultaat:=Antwoord/Resultaat;

end;



case jaar of

  Min..1939: begin

               Tijd:=VoorWO2;

               WriteLn ('De wereld in vrede..');

             end;

  1946..Max: begin

               Tijd:=NaWO2;
```

```
              WriteLn('De wereld wordt opgebouwd.');
            end;
        else begin
              Tijd:=WO2;
              WriteLn ('Oorlogstijd..');
            end;
end;
```

6.2.3 Lus opdracht

Lus opdrachten geven aan, dat bepaalde opdrachten herhaald uitgevoerd moeten worden.
Als het aantal herhalingen vooraf bekend is, dat betekend voordat de herhaling
plaatsvindt, is de for opdracht de geÎigende opdracht om deze situatie uit te drukken.
Anders zou de while of de repeat opdracht gebruikt moeten worden.

6.2.3.1 For opdracht

De for opdracht geeft aan, dat de deelopdracht herhaald uitgevoerd moet worden. De
oplopende waarden worden aan een variabele toegewezen, die de controlevariabele
genoemd wordt. De waarden kunnen oplopen: to of aflopend: downto tot de eindwaarde
is bereikt.

De controlevariabele, de begin- en de eindwaarde moeten van hetzelfde type zijn.
Geldige typen zijn alle enkelvoudige typen, dat betekend alle scalaire typen behalve Real.
Als bij de toepassing van de to de beginwaarde groter is dan de eindwaarde, of bij de
toepassing van downto de beginwaarde kleiner is als de eindwaarde, wordt de
deelopdracht niet uitgevoerd.

Voorbeelden:

```
for I:=2 to 100 do
  if A[I]>Max then Max:=A[I];

for I:=1 to AantalRegels do
```

```
begin

  ReadLn (Regel);

  if Length (Regel)<Limiet then

    KorteRegels:=KorteRegels+1

  else

    LangeRegels:=LangeRegels+1

end;
```

Let erop dat de deelopdracht van een for opdracht geen toewijzing aan de controlevariabele mag bevatten. Als de herhaling gestaakt moet worden voor de
eindwaarde bereikt is, moet een goto opdracht gebruikt worden, hoewel zulke
programmastructuren niet worden aangeraden. Het programma blijft duidelijker als in
zo'n geval een while of een repeat opdracht gebruikt wordt.

Na de beÎindiging van de for opdracht is de controlevariabele gelijk aan de eindwaarde,
behalve als de opdracht niet wordt uitgevoerd, dan wordt de controlevariabele niet
toegewezen.

6.2.3.2 While opdracht

De expressie, die de herhaling controleert, moet van het type Boolean zijn. De opdracht
wordt herhaald zolang de expressie True is. Is de waarde al bij het begin False dan wordt
de opdracht helemaal niet uitgevoerd.

Voorbeelden:

```
while Maat>1 do

  Maat:=Sqrt (Maat);



while DezeMaand do

begin

  DezeMaand:= NuMaand=VoorbldMaand;



  Verwerk;  {Bewerk dit voorbeeld met de Verwerk procedure}
```

```
end;
```

6.2.3.3 Repeat opdracht

De expressie, die de herhaling controleert, moet van het type Boolean zijn. De opdrachten
sequentie die tussen de gereserveerde woorden repeat en until staat wordt zo vaak
herhaald, tot de expressie waar (True) wordt. Het onderscheid met de while opdracht is
dat de repeat opdracht minstens ÈÈn keer wordt uitgevoerd, omdat pas aan het einde de
afbreekvoorwaarde gecontroleerd wordt.

Voorbeeld:

```
repeat

  Write (^M, 'Dit record verwijderen? (J/N)');

  Read (Antwoord);

until UpCase (Antwoord) in ['J', 'N'];
```

7. SCALAIRE TYPEN EN HUN DEELBEREIKEN

Het scalaire data type is bij Pascal het basis data type. Het vormt een eindige, lineair
geordende rij van waarden. Hoewel het standaard type Real ook als scalair data type
gezien wordt, valt het type Real niet onder deze definitie. Daarom kunnen Real getallen
niet altijd in gelijke samenhang met andere scalaire data typen gebruikt worden.

7.1 Scalaire typen

Naast de scalaire standaard typen (Longint, Integer, Real, Boolean, Char en Byte)
ondersteund Pascal ook door de gebruiker gedefinieerde scalaire typen. De definitie van
een scalair data type geeft in lineaire volgorde al zijn mogelijke waarden aan. De waarden
van het nieuwe data type worden door namen gerepresenteerd, die hun constanten zijn.

Voorbeelden:

```
type

  Operator = (Plus, Min, Maal, Deel);
```

```
  Dag      = (Maan, Dins, Woens, Donder, Vrij, Zater, Zon);

  Maand    = (Jan, Feb, Maart, Apr, Mei, Juni, Juli,

             Aug, Sep, Okt, Nov, Dec);

  Kaart    = (Schoppen, Klaver, Harten, Ruiten);
```

Variabelen van het type Kaart kunnen ÈÈn van de vier waarden (schoppen, klaver, harten of ruiten) aannemen. De ordinale waarden zijn:

Schoppen=0, Klaver=1, Harten=2, Ruiten=3.

Met het scalaire standaard type Boolean bent u al vertrouwd. Het is als volgt gedefinieerd:

```
type

  Boolean = (False, True);
```

De relationele operatoren =, <>, >, <, >= en <= kunnen worden gebruikt bij alle scalaire typen als beide operanden maar van hetzelfde type zijn (Reals, Integers en Longints mogen wel door elkaar gebruikt worden). De volgorde van het scalaire type wordt gebruikt als basis bij vergelijkingen, met andere woorden, de volgorde van de namenlijst is bepalend voor de ordinale waarde van een scalair type. In het bovenstaande voorbeeld is de volgende expressie waar (True):

```
Schoppen < Klaver < Harten < Ruiten
```

De volgende standaard functies kunnen worden gebruikt met als parameter een scalair type:

```
Succ (Klaver)   De opvolger van Klaver (Harten)

Pred (Klaver)   De voorganger van Klaver (Schoppen)

Ord (Klaver)    De ordinale waarde van Klaver (1, omdat het eerste element

                uit de scalaire lijst de waarde 0 heeft.)
```

De functies Succ en Pred hebben als resultaat hetzelfde type als hun parameter. De
functie Ord heeft als resultaat type een Integer.

7.2 Deelgebied typen

Een type kan worden gedefinieerd als een deelgebied van al eerder gedefinieerd scalair
type. We spreken dan van een deelgebied type. De definitie van een deelgebied bestaat
eenvoudig uit het specificeren van de laagste en de hoogste waarde van het deelgebied.
De eerste waarde moet kleiner zijn dan de tweede waarde. Een deelgebied van het type
Real is niet toegestaan.

Voorbeelden:

```
type

  Windstreken = (Noord, Zuid, Oost, West);

  Wereld      = (Oost..West);

  KompasRoos  = 0..360;

  Kapitalen   = 'A'..'Z';

  Onderkast   = 'a'..'z';

  Graden      = (Celsius, Fahrenheit, Kelvin);

  Wijn        = (Rood, Wit, RosÈ, Mousserend);
```

Het type Wereld is een deelgebied type van het type Windstreken (het ermee
geassocieerde type). Het geassocieerde scalaire type van KompasRoos is Integer, en het
geassocieerde scalair type van Kapitalen en Onderkast is het type Char.

We hebben het al eerder gehad over het type Byte. Dit is een standaard deelgebied type
welke is gedefinieerd als:

```
type

  Byte = 0..255;
```

Een deelgebied type heeft alle eigenschappen van zijn geassocieerd scalair type, maar is
beperkt tot zijn eigen minimale en maximale bereik.

Het gebruik van gedefinieerde scalaire typen en deelgebied typen wordt sterk
aanbevolen, omdat het de leesbaarheid van een programma sterk verhoogd. Daarnaast is
het mogelijk om de bereikcontrole (zie hoofdstuk 7.4) aan te zetten en te controleren of
een berekening binnen de gestelde bereiken blijft. Een ander voordeel van gedefinieerde
typen en hun deelgebied typen is dat het vaak minder geheugen inneemt. Turbo Pascal zal
maar 1 byte geheugen reserveren als het totaal aantal elementen uit een deelgebied type
minder dan 256 elementen bevat. M.a.w. als een Integer deelgebied type een boven en
ondergrens hebben binnen het gebied 0 t/m 255, zal er dus maar 1 byte geheugen voor
worden gereserveerd.

7.3 Type omzetten

De functie Ord kan worden gebruikt om een scalair type om te zetten naar een waarde
van het Integer type. Standaard Pascal voorziet niet in een mogelijkheid om dit proces om
te draaien, dus om een Integer waarde om te zetten in een scalair type.

In Turbo Pascal kan een scalair type worden omgezet naar een ander scalair type met
dezelfde ordinale waarde met behulp van type conversie (Engl.: type casting). Type
conversie wordt bereikt door de type naam van het gewenste type te nemen en deze als
een functie te gebruiken met als parameter de te converteren waarde. De parameter mag
van elk type zijn behalve Real.

Als we uitgaan van de type in hoofdstuk 7.1 en 7.2 dan geldt het volgende:

```
Integer (Harten) = 2

Maand (10)       = Nov

Windstreken (2)  = Oost

Kapitalen (14)   = 'O'

Graden (2)       = Kelvin

Char (78)        = 'N'

Integer ('7')    = 55
```

7.4 Bereik controle

De code generatie die er voor zorgt dat er tijdens het lopen van een programma op het bereik wordt gecontroleerd, wordt bepaald door de R compiler aanwijzing. Standaard staat deze controle uit {$R-}, er wordt dus niet op gecontroleerd. Als er een toewijzing plaats vindt terwijl deze controle aan staat {$R+} zal worden gecontroleerd of de toewijzing binnen het gestelde bereik valt. Het wordt aangeraden om deze controle bij het ontwikkelen van een programma aan te zetten totdat het programma volledig foutloos is (zie hoofdstuk 9.1 voor de invloed van deze compiler aanwijzing op arrays).

Voorbeeld:

```
program bereikcontrole;

type

  Cijfer = 0..9;

var

  Cijfer1, Cijfer2, Cijfer3 : Cijfer;

begin

  Cijfer1:=5;          {Geldig}

  Cijfer2:=Cijfer1+3; {Geldig omdat Cijfer1+3<=9}

  Cijfer3:=47;         {Ongeldig maar er treedt geen fout op}

  {$R+}
```

```
  Cijfer3:=55;        {Ongeldig en veroorzaakt een melding}

  {$R-}

  Cijfer3:=167;       {Ongeldig maar er treedt geen fout op}

end.
```

8. STRING TYPE

Turbo Pascal biedt u het data type String, om tekenreeksen te verwerken. Tekenreeksen
zijn aaneenschakelingen van tekens. Het type String is gestructureerd en lijkt veel op het
array (hoofdstuk 9). Er is echter een groot verschil: het aantal tekens in een String en zijn
lengte kan dynamisch variÎren tussen 0 en een bovengrens, terwijl het aantal elementen
in een array vaststaat.

8.1 String type definitie

De definitie van het data type String moet de bovengrens van het aantal te bevatten
tekens, dat is de maximale lengte, aangeven. De definitie bestaat uit het gereserveerde
woord string, waarop tussen blokhaken de maximale lengte volgt. Deze moet een Integer
constante zijn varieerend van 1 t/m 255. Strings hebben geen vooringestelde lengte en
daarom moet de lengte precies bepaald worden.

Voorbeeld:

```
type

  FileName   = string[63];

  ScreenLine = string[80];
```

Een String variabele gebruikt in het geheugen zijn maximale lengte plus 1 byte. Dit ene
byte wordt gebruikt om de actuele lengte van de String variabele in op te slaan. Alle
tekens in de String variabele zijn genummerd, beginnend met 1 t/m de lengte van de
String. Index 0 bevat de lengte.

8.2 String expressies

Strings worden door String expressies bewerkt. Deze bestaan uit String constanten, String
variabelen, functienamen en operaties.

Het plusteken kan Strings verbinden. De functie Concat (zie hoofdstuk 8.5.2) doet
hetzelfde, maar de '+' operator is vaak eenvoudiger te verwerken. Zou de lengte van de
ontstaande String groter dan 255 zijn, wordt een looptijdfout gegeven.

Voorbeelden:

'Turbo ' + 'Pascal'   = 'Turbo Pascal'

'123' + '.' + '456'   = '123.456'

'A' + 'B' + 'C' + 'D' = 'ABCD'

De relationele operatoren =, <, >, <= en >= hebben een lagere prioriteit dan de
verbindingsoperator. Als relationele operatoren op String operanden toegepast worden is
het resultaat van het type Boolean. Bij de vergelijking van twee Strings worden de aparte
letters van links naar rechts met elkaar vergeleken. Als de Strings verschillende lengten
hebben en de kortere tot de laatste letter gelijk is aan de aan het begin staande letters van
de langere String, dan wordt de kortere als kleiner erkend. Strings zijn alleen dan gelijk,
als ze zowel in lengte als inhoudelijk hetzelfde zijn.

Voorbeelden:

'A' < 'B'	waar
'A' > 'B'	niet waar
'1' < '12'	waar
'2' < '12'	niet waar
'TURBO' = 'TURBO'	waar
'TURBO ' = 'TURBO'	niet waar
'Pascal Compiler' < 'Pascal compiler'	waar

8.3 String toewijzing

De toewijzingsoperator wijst de waarde van een String expressie aan een String variabele
toe.

Voorbeelden:

```
Leeftijd:=' vijftiende';

Regel   :='Gefeliciteerd met je' + Leeftijd + 'verjaardag.';
```

Als de aangegeven maximale lengte van een String overschreden wordt, worden de
overtollige letters weggegooid. Dat betekend dat als de bovenstaande variabele Leeftijd
gedeclareerd zou zijn als string[5], dan bevat de variabele na de toewijzing maar vijf
letters. De waarde zou dan dus worden : ' vijf'.

8.4 String procedures

De volgende standaard procedures voor Strings zijn in Turbo Pascal beschikbaar:

8.4.1 Delete

Syntax: Delete (St, Pos, Num)

Delete wist uit een String variabele (St) een bepaald aantal (Num) letters, beginnend op de
positie Pos. St is een String variabele en zowel Pos als Num zijn Integer expressies. Als
Pos groter is dan de lengte van St, worden er geen letters verwijderd. Als geprobeerd
wordt letters te wissen die zich buiten het rechtereinde van de String bevinden, dus als
Pos+Num groter is dan de lengte van St, worden alleen tekens binnen St gewist. Als Pos
buiten het bereik 0 t/m 255 ligt, wordt een looptijdfout gegeven.

Als St de waarde 'ABCDEFG' heeft, dan neemt St onder de volgende voorwaarden de
nevenstaande waarden aan:

```
Delete (St, 2, 4)        geeft een waarde van 'AFG'

Delete (St, 2, 10)       geeft een waarde van 'A'

Delete (St, 10, 1)       doet niets!!
```

8.4.2 Insert

Syntax: Insert (Obj, Target, Pos)

Insert voegt de String Obj in de String Target op positie Pos in. Obj is een String
expressie, Target een String variabele en Pos een Integer getal. Als Pos groter is dan de

lengte van de String variabele Target, dan wordt Obj achter aan Target toegevoegd. Als
het resultaat langer is dan de maximale lengte van de variabele Target, verdwijnen de
overtollige letters en Target bevat slechts de meest links staande letters van Obj. Als Pos
buiten het bereik van 0 t/m 255 ligt, wordt een looptijdfout gegeven.

Als St de waarde 'ABCDEFG' heeft dan neemt St onder de volgende voorwaarde de
nevenstaande waarde aan:

Insert ('XX', St, 3)   geeft een waarde: 'ABXXCDEFG'.

8.4.3 Str

Syntax: Str (Value, Str)

De procedure Str converteert de numerieke waarde Value in een String en slaat die in St
op. Value is een schrijfparameter van het type Real, Longint of Integer, St is een String
variabele. Schrijfparameters zijn expressies met speciale formateer commando's (zie
hoofdstuk 13.11.1).

Als I de waarde 1234 heeft, geldt:

Str (I:5, St)   St krijgt de waarde ' 1234'

Als X de waarde 2.5E4 heeft, geldt:

Str (X:10:0, St)   St krijgt de waarde ' 25000'

Waarschuwing: Een functie waarin de Str procedure voorkomt mag nooit gebruikt
worden in een expressie die binnen een Write of WriteLn staat.

8.4.4 Val

Syntax: Val (St, Var, Code)

Val converteert de String expressie St in een Longint, Integer of Real waarde (afhankelijk

van het type van de variabele Var) en slaat deze op in de variabele Var. St moet een String zijn, die een numerieke waarde voorstelt, overeenkomend met de regels voor numerieke constante (zie hoofdstuk 3.2). Noch ervoor, noch erna zijn lege tekens toegestaan. Var moet een Integer of Real variabele zijn. Als geen fouten gevonden worden, wordt de variabele Code op 0 gezet, anders bevat Code de positie van het eerste foutieve teken, en blijft de waarde van Var ongedefinieerd.

Als St de waarde '234' heeft, geldt:

Val (St, I, Result)   I krijgt de waarde 234 en Result wordt 0

Als St de waarde '12x' heeft, geldt:

Val (St, I, Result)   I blijft ongedefinieerd en Result wordt 3

Als St de waarde '2.5E4' heeft, en X een real variabele is, geldt:

Val (St, X, Result)   X krijgt de waarde 25000 en Result wordt 0

Waarschuwing: Een functie die de Val procedure gebruikt mag nooit in een expressie in een Write of WriteLn aangeroepen worden.

8.5 String functies

De volgende standaard functies voor Strings zijn in Turbo Pascal toepasbaar:

8.5.1 Copy

Syntax: Copy (St, Pos, Num)

Copy geeft een deel van de String St terug, die een bepaald aantal (Num) letters bevat, geteld van de positie Pos. St is een String expressie, Pos en Num zijn Integer expressies.
Als de waarde van Pos boven de lengte van de String ligt, wordt een lege String

teruggegeven. Als geprobeerd wordt tekens rechts buiten de String te verkrijgen, dus als
Pos+Num groter is als de lengte van de String, worden alleen de letters die binnen de
String liggen teruggegeven. Als Pos buiten het bereik van 0 t/m 255 ligt wordt een
looptijdfout gegeven.

Als St de waarde 'ABCDEFG' heeft, geldt:

```
Copy (St, 3, 2)            geeft de waarde 'CD' terug

Copy (St, 4, 10)           geeft de waarde 'DEFG' terug

Copy (St, 4, 2)            geeft de waarde 'DE' terug
```

8.5.2 Concat

Syntax: Concat (St1, St2[, StN])

De functie Concat geeft een verzamel String terug, die uit een willekeurig aantal deel
Strings in de aangegeven volgorde (St1..StN) samengesteld wordt. Als het resultaat groter
is dan 255 tekens, wordt een looptijdfout gegeven. Zoals in hoofdstuk 8.2 reeds
aangegeven, kan de '+' operator hetzelfde en dat op een eenvoudigere manier. Concat
werd alleen in Turbo Pascal opgenomen om de compatibiliteit met andere Pascal
compilers te behouden.

Als St1 de waarde 'TURBO' en St2 de waarde ' is de snelste' heeft, geldt:

```
Concat (St1, ' Pascal', St2)  geeft de waarde 'Turbo Pascal is de snelste'
```

8.5.3 Length

Syntax: Length (St)

Geeft de lengte van de String expressie St terug, dit is het aantal tekens dat de String
bevat. Het resultaat is een Integer.

Als St de waarde '123456789' heeft, dan geeft:

Length (St) de waarde 9.

8.5.4 Pos

Syntax: Pos (Obj, Target)

Deze functie doorzoekt de String Target naar het eerste voorkomen van de String
expressie Obj. Het resultaat is Integer en geeft de positie in de String expressie Target
aan die het eerste teken van Obj heeft. De eerste positie in een String is 1. Als de
tekencombinatie Obj niet gevonden wordt, levert Pos de waarde 0.

Als St de waarde 'ABCDEFG' heeft, dan geeft:

```
Pos ('DE', St)              de waarde 4

Pos ('H', St)               de waarde 0
```

8.6 Strings en tekens

Het data type String en het scalaire data type Char zijn uitwisselbaar. Zo kan, waar een
String waarde verwacht wordt, ook een tekenwaarde ingevoerd worden en omgekeerd.
Buiten dat, kunnen Strings en tekens ook in expressies gemengd worden. Als een teken
naar een String geconverteerd wordt moet de lengte als '1' gedefinieerd worden anders
wordt een looptijdfout gegeven. Op de tekens van een String kan men toegang krijgen
door String indexering. Daarvoor wordt aan de String variabele een Integer getal tussen
blokhaken toegevoegd.

Voorbeelden:

```
Buffer[5]

Line [Length (Line)-1]

Ord (Line [0])
```

Omdat het eerste teken van een String (bij index '0') de lengte van de String bevat, is
Length(String) hetzelfde als Ord(String[0]) Als de lengte indicator gebruikt

wordt moet de programmeur erop letten dat deze korter is dan de maximale lengte van de String variabele. Als de compiler aanwijzing R actief is ({$R+}), wordt de code zo gegenereerd dat gegarandeerd wordt dat een String index expressie zijn maximale lengte niet zal overschrijden. Het is desondanks altijd nog mogelijk een String index te vormen die boven de actuele dynamische lengte ligt. De op die manier gelezen tekens zijn ongedefinieerd en toewijzingen voorbij de actuele dynamische lengte veranderen de String niet (het systeem kan hierop zelfs vastlopen).

9. ARRAY TYPE

Een array is een gestructureerd data type met een vastgesteld aantal elementen, die allemaal van hetzelfde type zijn, het basis type. Op elk element kan toegang verkregen worden door gebruik te maken van indices. Een index is een Integer expressie die tussen blokhaken achter de array naam staat. Zijn data type wordt index type genoemd.

9.1 Array definitie

De definitie van een array bestaat uit het gereserveerde woord array waarop het index type tussen blokhaken volgt. Daarna staat het gereserveerde woord of gevolgd door het basis type.

Voorbeelden:

```
type

  Dag=(Maan, Dins, Woens, Donder, Vrij, Zater, Zon);

var

  Werkuren : array[1..8] of Integer;

  Week     : array[1..7] of Dag;



type

  Spelers= (Speler, Speler2, Speler3, Speler4);

  Hand   = (One, Two, Pair, TwoPair, Three, Straight,

            Flush, FullHouse, Four, StraightFlush, RSF);

  GoedBod= 1..200;
```

```
  Bod    = array[Spelers] of GoedBod;

var

  Speler : array[Spelers] of Hand;

  Pot    : Bod;
```

Op een array element wordt toegang verkregen door aan de variabele naam een index
tussen blokhaken te hangen:

```
Speler[Speler3]:=FullHouse;

Pot[Speler3]   :=100;

Speler[Speler4]:=Flush;

Pot[Speler4]   :=50;
```

Omdat toewijzingen tussen twee willekeurige variabelen toegestaan is, mits ze van
hetzelfde type zijn, kunnen hele arrays met behulp van ÈÈn toewijzing gecopieerd
worden.

De compiler aanwijzing R controleert bij de codegeneratie of de array indexeringen
binnen het bereik liggen. De default instelling is inactief. De instelling {$R+}
veroorzaakt een controle van alle array indexeringen op het feit of ze binnen de grenzen
vallen.

9.2 Meerdimensionale arrays

De elementen van een array kunnen van een willekeurig data type zijn, dat betekend dat
de elementen ook arrays mogen zijn. Zo'n structuur noemt men meerdimensionale arrays.

Voorbeeld:

```
type

  Kaart =( Twee, Drie, Vier, Vijf, Zes, Zeven, Acht,
```

```
         Negen, Tien, Boer, Vrouw, Koning, Aas );

  Troef =( Harten, Schoppen, Klaveren, Ruiten );

  AlleKaarten = array[Troef] of array [1..13] of Kaart;

var

  Dek : AlleKaarten;
```

Het meerdimensionale array kan ook eenvoudiger gedefinieerd worden:

```
type

 AlleKaarten = array[Troef, 1..13] of Kaart;
```

Een dergelijk kortschrift is ook van toepassing op verwijzingen naar array elementen:

```
Dek[Harten,10] is hetzelfde als Dek[Harten][10]
```

Het is natuurlijk ook mogelijk om meerdimensionale arrays van eerder gedefinieerde
array typen te maken.

Voorbeeld:

```
type

  Leerlingen = string[20];

  Klas       = array[1..30] of Leerlingen;

  School     = array[1..50] of klas;

var

  J, P, Vrij  : Integer;

  KlasA,KlasB : Klas;

  MariaSchool : School;
```

Na deze definitie zijn de volgende toewijzingen mogelijk:

```
KlasA[J]          :='Peter';

MariaSchool[5][21]:='Peter Brown';

MariaSchool[8,J]  := MariaSchool[7,J]; {leerling J wisselt van klas}

KlasA[Vrij]       := KlasB[P];  {leerling P wisselt van klas en nummer}
```

9.3 Teken arrays

Teken arrays zijn arrays met elementen van het scalaire data type Char. Teken arrays
kunnen als Strings met een constante lengte gezien worden.

Bij Turbo Pascal kunnen teken arrays in String expressies voorkomen. In dat geval wordt
de array in een String van de desbetreffende lengte geconverteerd. Zo kunnen arrays en
Strings op dezelfde manier vergeleken en behandeld worden en String constanten kunnen
teken arrays toegewezen krijgen, zolang ze dezelfde lengte hebben. String variabelen en
waarden uit String expressies kunnen niet aan teken arrays toegewezen worden.

9.4 Voorgedefinieerde arrays

Turbo Pascal kent twee voor gedefinieerde arrays van het type Byte namelijk: Mem en
Port. Deze dienen als directe toegang tot het CPU-geheugen en tot de data poorten.

9.4.1 Mem array

Het voor gedefinieerde array Mem wordt gebruikt om direct toegang te krijgen tot het
geheugen. Als er een waarde wordt toegekend aan een element van Mem dan wordt deze
waarde direct in de overeenkomstige geheugenplaats ingevuld. Een gelezen waarde komt
direct uit de bijbehorende geheugenplaats.

9.4.2 Port array

Het voorgedefinieerde array Port wordt gebruikt om direct toegang te krijgen tot de data
poorten van de computer. Als er een waarde wordt toegekend aan een element van Port
dan wordt deze waarde direct naar de overeenkomstige poort gestuurd. Een gelezen
waarde wordt eerst van de bijbehorende poort opgehaald.

Bij het Port array geldt de beperking dat deze niet zonder een index mag worden gebruikt. Het is dus niet mogelijk alle poort waarden in ÈÈn toekenning aan een ander array toe te wijzen.

10. RECORD TYPE

Een record is een data type, welke uit een vaststaand aantal elementen bestaat, die velden worden genoemd. De velden kunnen uit verschillende data typen bestaan en elk veld wordt met een veldnaam (field identifier) benoemd. Deze dient voor de veldselectie in een record.

10.1 Record definitie

De definitie van het data type record bestaat uit het gereserveerde woord record, waarop een lijst van de aparte velden volgt (de field list). Daarna komt het gereserveerde woord end. De veldenlijst is een opeenvolging van regels (record sections), die door puntkomma's gescheiden zijn. Elke regel bestaat uit ÈÈn of meer veldnamen, gevolgd door een dubbele punt en een datatypenaam of een datatypeomschrijving. Zo bepaald elke regel het type en de naam van ÈÈn of meer velden.

Voorbeelden:

```
type
  Datum=record
          Dag   : 1..31;
          Maand :( Jan, Feb, Maart, April, Mei, Juni,
                   Juli, Aug, Sep, Okt, Nov, Dec );
          Jaar  : 1900..1999;
        end;
var
  Geboorte : Datum;
  WerkDag : array[1..5] of Datum;
```

Dag, Maand en Jaar zijn veldnamen. Een veldnaam is alleen geldig binnen het record
waarin het gedefinieerd wordt. Een veld wordt door middel van een punt van de variabele
naam gescheiden.

Voorbeelden:

```
Geboorte.Maand  := Juni;

Geboorte.Jaar   := 1950;

WerkDag[Huidige]:= WerkDag[Huidige-1];
```

Let erop dat, net als bij arrays, een toewijzing tussen gehele records van hetzelfde type
mogelijk is. Omdat de aparte elementen willekeurige data typen mogen bezitten, zijn de
volgende constructies van in elkaar schakeling mogelijk:

```
type

  Naam = record

           FamilieNaam : string[32];

           DoopNaam    : array[1..3] of string[16];

         end;

  Uren = record

           NormaalWerk, OverWerk,

           NachtWerk, Weekend : Integer;

         end;

  Datum = record

            Dag   : 1..31;

            Maand : (Jan, Feb, Maart, April, Mei, Juni,

                     Juli, Aug, Sep, Okt, Nov, Dec);

            Jaar  : 1900..1999;

          end;

  Persoon = record
```

```
            ID   : Naam;

            Tijd : Datum;

          end;

  Loon = record

         Werknemer : Persoon;

         Kost      : Uren;

       end;


var Salaris, Beloning : Loon;
```

Na dit voorbeeld zouden de volgende toewijzingen mogelijk zijn:

```
Salaris := Beloning;

Salaris.Kost.OverWerk := 950;

Salaris.Werknemer.Tijd := Beloning.Werknemer.Tijd

Salaris.Werknemer.ID.FamilieNaam := 'Jansen';
```

10.2 With opdracht

Het gebruik van records zoals dat hiervoor is beschreven kan leiden tot erg lange opdrachten. Het zou vaak handiger zijn als elk veld benaderd kan worden met alleen zijn veldnaam. Dit is mogelijk met de with opdracht. Het 'opent' een record op zo'n manier dat alle veldnamen kunnen worden gebruikt als een variabele naam

De with opdracht bestaat uit het gereserveerde woord with, gevolgd door een lijst met record variabelen gescheiden door komma's en gevolgd door het gereserveerde woord do en tot slot een opdracht.

Binnen de with opdracht kan het veld benaderd worden door alleen de veldnaam te gebruiken. De recordnaam is dan dus niet meer nodig.

```
with Salaris do

begin

  Werknemer := Nieuwkomer;

  Kost      := StandaardUren;

end;
```

Records kunnen worden genest binnen with opdrachten, dus records van records kunnen
worden 'geopend' zoals hieronder:

```
{$W3}

with Salaris, Werknemer, ID do

begin

  FamilieNaam := 'Jansen';

  DoopNaam[1] := 'Jozef';

end;
```

Dit is gelijk aan:

```
  with Salaris do with Werknemer do with ID do

  ...
```

De maximale diepte van de nesting van de with opdracht is in te stellen met de W
compiler opdracht. Standaard is de diepte 2 {$W2}. Dit kan echter per versie van Turbo
Pascal verschillend zijn.

10.3 VariÎrende records

De schrijfwijze van het record type voorziet ook in variÎrende records, met andere
woorden een record structuur waar binnen een record de velden kunnen veranderen,
meestal afhankelijk van de waarde van een keuzeveld.

Laten we aannemen dat het volgende type bestaat:

```
Afkomst = (Plaatselijk, VanElders);
```

en het type Naam en Datum zoals in hoofdstuk 10.1. Het volgende record heeft andere velden afhankelijk of de waarde van AfkomstPlaats, Plaatselijk of VanElders is.

```
type
  Persoon=record
            Voornaam:Naam;
            GeboorteDatum:Datum
            case AfkomstPlaats : Afkomst of
              Plaatselijk : (GeboortePlaats  :Naam);
              VanElders:(GeboorteLand        :Naam;
                         DatumInschrijving   :Datum;
                         VerguningTot        :Datum;
                         PlaatsVanBinnenkomst:Datum);
            end;
```

In deze variÎrende record definitie kan een keuze veld op dezelfde manier bewerkt worden als normale velden. Dus als Passagier van het type Persoon is zijn de volgende opdrachten toegestaan:

```
Passagier.AfkomstPlaats:=Plaatselijk;
with Passagier, Voornaam do
  if AfkomstPlaats=VanElders then Write(Familienaam);
```

Het vaststaande deel van een record, het deel dat de normale velden bevat, moet altijd voorafgaan aan het variÎrende deel. In het bovenstaande voorbeeld zijn de velden

Voornaam en GeboorteDatum de vaststaande velden. In het variÎrende deel moeten de
veldnamen tussen ronde haken staan, ook al staat er niets binnen deze haken.

Het bijhouden en bewerken van het keuze veld is een verantwoordelijkheid van de
programmeur en niet van Turbo Pascal. In het type Persoon kan het veld DatumInschrijving worden benaderd, ook als in het keuzeveld AfkomstPlaats de waarde
Plaatselijk staat in plaats van VanElders.

De waarde van het keuze veld AfkomstPlaats kan volledig worden genegeerd zodat alleen
de type naam overblijft. Dit wordt het werken met vrije secties (free unions) genoemd.
Het gebruik maken van het keuze veld heet werken met beperkte secties (discriminated
unions). Het gebruik van vrije secties is erg ongebruikelijk en zou alleen gebruikt moeten
worden door zeer ervaren programmeurs.

11. SETS

Een verzameling (set) is een samenvatting van meerdere objecten van hetzelfde type, die
als geheel gezien worden. Elk apart object van zo'n verzameling wordt een element
genoemd. Enkele voorbeelden:

1) Alle integer getallen tussen 1 en 100

2) Alle letters van het alfabet

3) Alle klinkers van het alfabet

Twee verzamelingen zijn alleen dan gelijk aan elkaar, als hun elementen hetzelfde zijn.
Er is geen rangorde, zodat de verzamelingen [1, 3, 5], [1, 5, 3] en [5, 3, 1] gelijk zijn. Als
de elementen van een verzameling ook de elementen van een andere verzameling zijn,
geldt dat de eerste verzameling een deelverzameling is van de tweede. (De tweede bevat
de eerste). In het bovenstaande voorbeeld is 3 een deelverzameling van 2.

Met verzamelingen kan men op drie verschillende manieren rekenen (net als optellen,

aftrekken en vermenigvuldigen met getallen):

De vereniging van twee verzamelingen A en B (geschreven als A+B) is de verzameling
waarvan de elementen in A of B voorkomen. De vereniging van [1, 3, 4, 5, 7] en [2, 3,
4] is [1, 2, 3, 4, 5, 7].

De doorsnede van twee verzamelingen (geschreven A*B) is de verzameling waarvan de
elementen in A en B voorkomen. De doorsnede van [1, 3, 4, 5, 7] en [2, 3, 4] is [3, 4].

Het verschil of het complement van twee verzamelingen, het relatieve complement,
(geschreven A-B) is de verzameling van elementen uit de eerste verzameling die niet in
de tweede voorkomen. Het verschil van [1, 3, 5, 7] en [2, 3, 4] is [1, 5, 7].

11.1 Set definitie

Hoewel er in de wiskunde geen beperkingen zijn welke elementen in een verzameling
kunnen voorkomen heeft Pascal zijn restricties. De elementen van een verzameling
moeten van hetzelfde type (het basis type) zijn en het moet een eenvoudig type zijn, dat
betekend een scalair data type, uitgezonderd de Real en de Longint. Een verzameling
wordt vooraf gegaan door het gereserveerde woord set of, gevolgd door een eenvoudig
data type.

Voorbeelden:

```
type

  DagenPerMaand = set of 0..31;

  WerkWeek      = set of Maan..Vrij;

  Letter        = set of 'A'..'Z';

  MengKleuren   = set of (Rood, Groen, Blauw);

  Karakters     = set of Char;
```

Bij Turbo Pascal kan een verzameling maar 256 elementen bevatten en de ordinale

getallen van het basis type moeten tussen de 0 en 255 liggen.

11.2 Set expressies

De waarden van een set kunnen met de waarden van een andere set door set expressies en set operaties verkregen worden. Set expressies bestaan uit set constanten, set variabelen, weergaven van sets en set operatoren.

11.2.1 Weergave van de set

De weergave van een verzameling bestaat uit ÈÈn of meer, door komma's gescheiden en door blokhaken omgeven element bepalingen. Een element bepaling is een expressie, van hetzelfde data type als het basis type van de verzameling, of het is een bereik, die door twee van zulke expressies gescheiden door twee opeenvolgende punten wordt bepaald.

Voorbeelden:

```
['T','U','R','B','O']

[X,Y]

[X..Y]

[1..5]

['A'..'Z','a'..'z','0'..'9']

[1,3..10,12]

[]
```

Het laatste voorbeeld laat de lege verzameling zien. Omdat de lege verzameling geen expressie bevat die haar basis type aangeeft is de lege verzameling compatibel is met alle set typen. De verzameling [1..5] is hetzelfde als de verzameling [1,2,3,4,5]. Als X>Y, dan staat [X..Z] voor een lege verzameling.

11.2.2 Set operatoren

De regels van verzamelingen geven de voorrang van de set operaties in drie verschillende klassen aan:

1) * Doorsnede van de verzamelingen.

2) + Vereniging van de verzamelingen.

- Verschil van de verzamelingen.

3) = Test op de gelijkheid van de verzamelingen.

<> Test op de ongelijkheid van de verzamelingen.

>= 'Bevat' is waar, als de eerste verzameling de tweede bevat.

<= 'Is bevat' is waar, als de tweede verzameling de eerste bevat.

IN Test of een element in een verzameling voorkomt. De eerste operand is een element met als type het basis type van de verzameling die de tweede operand vormt. Het resultaat is True als de eerste operand een element is van de tweede.

Er is geen operator voor de exclusie, maar men kan hem als volgt programmeren:

```
A * B = []
```

Set expressies kunnen voor de vereenvoudiging van gecompliceerde tests zeer behulpzaam zijn:

```
  if (Ch='T')or(Ch='U')or(Ch='R')or(Ch='B')or(Ch='O')
```

kan ook duidelijker uitgedrukt worden:

```
  if Ch in ['T','U','R','B','O']
```

En de test:

```
  if (Ch>='0') and (Ch<='9') then ...
```

zou er als volgt beter uitzien:

```
  if Ch in ['0'..'9'] then ...
```

11.3 Set toewijzingen

Waarden die uit set expressies ontstaan worden aan set variabelen middels de toewijzing
operator := worden toegewezen.

Voorbeelden:

```
type

  ASCII = set of 0..127;

var

  NoPrint, Print, AllChars : ASCII;

begin

  AllChars := [0..127];

  NoPrint  := [0..31, 127];

  Print    := AllChars - NoPrint;

end;
```

12.   GETYPEERDE CONSTANTEN

Getypeerde constanten zijn een specialiteit van Turbo Pascal. Een getypeerde constante
mag precies hetzelfde gebruikt worden als een variabele van hetzelfde type. Getypeerde
constanten kunnen dus worden gezien als geÔnitialiseerde variabelen omdat de waarde
van een getypeerde constante vast ligt. De waarde van een variabele is niet gedefinieerd
totdat er een eerste toewijzing aan heeft plaatsgehad. Er moet natuurlijk wel op gelet
worden dat er geen toewijzingen plaatsvinden aan constanten die daadwerkelijk als
constant bedoeld zijn.

Het gebruik van getypeerde constanten bespaart code als de constante vaak in een
programma wordt gebruikt. Dit komt omdat een getypeerde constante maar ÈÈn keer in de

code wordt opgenomen en een niet getypeerde constante iedere keer wordt ingevuld als
de constante gebruikt wordt.

Een getypeerde constante wordt op dezelfde manier gedefinieerd als een niet getypeerde
constante, maar er wordt behalve de waarde van de constante ook het type opgegeven. In
de definitie wordt de getypeerde constante naam voorafgegaan door een dubbele punt en
een typenaam, welke wordt gevolgd door een is gelijk teken (=) en de waarde van de
constante.

12.1 Ongestructureerde getypeerde constanten

Een niet gestructureerde getypeerde constante is een constante die gedefinieerd is als ÈÈn
van de scalaire typen:

```
const

  AantalAutos : Integer  = 1267;

  Rente       : Real     = 12.67;

  Codering    : String[6]= 'SECTIE';

  Xon         : Char     = ^Q;

  Naam        : Longint  = 6550000;
```

In tegenstelling tot niet getypeerde constanten kan een getypeerde constante worden
gebruikt in plaats van een variabele parameter naar een procedure of functie.

Omdat een getypeerde constante in werkelijkheid een variabele met een constante waarde
is, mag een getypeerde constante niet gebruikt worden voor het definiÎren van andere
constanten of typen. Als Min en Max getypeerde constanten zijn dan is de volgende
constructie niet toegestaan:

```
const

  Min: Integer = 0;

  Max: Integer = 50;
```

```
type

  Bereik: array[Min..Max] of Integer;
```

12.2 Gestructureerde getypeerde constanten

Gestructureerde constanten omvatten array constanten, record constanten en set constanten. Deze constanten worden vaak gebruikt voor het maken van tabellen en sets die al van een waarde zijn voorzien voor testdoeleinden, conversies, enz.. In de volgende paragrafen wordt elk type in detail besproken.

12.2.1 Array constanten

De definitie van een array constante bestaat uit de naam van de constante gevolgd door een dubbele punt en de type naam van een vooraf gedefinieerd type, die weer gevolgd wordt door een is gelijk teken (=) en een constante expressie omgeven door ronde haken en gescheiden door komma's.

Voorbeeld:

```
type

  Status    =(Actief, Passief, Wachtstand);

  StringPres=array[status] of String[10];

const

  Stat: StringPres=('Actief','Passief','Wachtstand');
```

Het voorbeeld definieert het constante array Stat, welke gebruikt kan worden om het scalaire type Status om te zetten in een overeenkomstige String presentatie. De elementen van Stat zijn:

```
  Stat[Actief]     = 'actief'

  Stat[Passief]    = 'passief'

  Stat[Wachtstand] = 'wachtstand'
```

De componenten van een constante array mogen van elk type zijn behalve van het File
type en het Pointer type. Arrays van karakters kunnen zowel als losse karakters
gedefinieerd worden, als doormiddel van een String. Dus de definitie:

```
const

  Cijfers: array[0..9] of Char =

           ('0','1','2','3','4','5','6','7','8','9');
```

kan gemakkelijker worden gedefinieerd als:

```
const

  Cijfers: array[0..9] of Char = ('0123456789');
```

12.2.2 Meerdimensionale array constanten

Meerdimensionale array constanten worden gedefinieerd door de constante van elke
dimensie tussen ronde haken en gescheiden door komma's te noteren. De constante die
het diepst naar binnen ligt komt overeen met de meest rechtse dimensie.

Voorbeeld:

```
type

  Kubus = array[0..1,0..1,0..1] of Integer;

const

  Doolhof : Kubus = (((0,1),(2,3)),((4,5),(6,7))):

begin

  WriteLn(Doolhof[0,0,0],' = 0');

  WriteLn(Doolhof[0,0,1],' = 1');

  WriteLn(Doolhof[0,1,0],' = 2');

  WriteLn(Doolhof[0,1,1],' = 3');

  WriteLn(Doolhof[1,0,0],' = 4');

  WriteLn(Doolhof[1,0,1],' = 5');
```

```
  WriteLn(Doolhof[1,1,0],' = 6');

  WriteLn(Doolhof[1,1,1],' = 7');

end.
```

12.2.3 Record constanten

De definitie van een record constante bestaat uit een constante naam gevolgd door een
dubbele punt en een eerder gedefinieerde type naam. Dit wordt weer gevolgd door een is
gelijk teken en een lijst veld constanten gescheiden door puntkomma's en ingesloten door
ronde haken.

Voorbeeld:

```
type

  Punt = record

           x,y,z : Integer;

         end;

  Systeem   = (MSX, CPM80, MSDOS, UNIX);

  Interface = (DosHelp, CCP, Norton, Klickit);

  Computer  = record

                OS  : array[1..4] of Systeem;

                GUI : Interface;

              end;

const

  Oorsprong: Punt     = (x:0; y:0; z:0);

  SuperComp: Computer = (OS:(DosHelp, CCP, Norton, Klickit);

                         GUI:DosHelp);

  Paneel: array[1..3] of Point = ((x:1; y:4; z:5),

                                  (x:10; y:-78; z:45),

                                  (x:100; y:10; z:-7));
```

De veld constanten moeten in dezelfde volgorde worden gespecificeerd als de volgorde in

de record definitie. Als er in een record een veld van het type Pointer voorkomt, dan kan dit record niet in een constante definitie worden opgenomen. Indien er een variÎrend record moet worden opgenomen, dan is het de verantwoordelijkheid van de programmeur om alleen geldige velden te specificeren. Als er een keuze veld aanwezig is, dan moet die waarde worden gespecificeerd.

12.2.4 Set constanten

Een set constante bestaat uit ÈÈn of meer elementen specificaties gescheiden door komma's en omgeven door rechte haken. Een element specificatie moet een constante zijn of een element bepaling bestaande uit twee constanten gescheiden door twee opvolgende punten (..).

Voorbeeld:

```
type

  Kapitalen = set of 'A'..'Z';

  Onderkast = set of 'a'..'z';

const

  HoofdLetters : Kapitalen   = ['A'..'Z'];

  Klinkers     : OnderKast   = ['a','e','i','o','u','y'];

  Scheiding    : set of Char = [#33..#47];
```

13. FILE TYPEN

Files voorzien een programma van kanalen om data te transporteren een file kan zowel een diskfile zijn als een logisch bestand. Bij een diskfile wordt een magnetisch opslagmedium benaderd en bij een logisch bestand voor gedefinieerde kanalen zoals Input en Output. Deze twee refereren aan de I/O kanalen van de computer: het toetsenbord en het beeldscherm.

Een file bestaat uit opeenvolgende componenten van hetzelfde type. Het aantal componenten in een file (de lengte van de file) wordt niet bepaald door de definitie van een file. In plaats hiervan houdt Pascal een file wijzer bij. Iedere keer als een component wordt weggeschreven of gelezen uit een file wijst deze wijzer naar het volgende element

in de file. Als alle componenten in de file dezelfde lengte hebben kan de positie van een
component worden berekend. In dit geval kunnen de componenten van de file in
willekeurige volgorde worden benaderd.

13.1 File type definitie

Een file wordt gedefinieerd met het gereserveerd woord file of gevolgd door het type van
de componenten in de file. Een file naam wordt gedefinieerd met dezelfde woorden
gevolgd door de naam van een hiervoor gedefinieerd File type.

Voorbeeld:

```
type

  ProductNaam= String[80];

  Product    = file of record

                    Naam       : ProductNaam;

                    ItemNummer : Real;

                    Voorraad   : Real;

                    MinVoorraad: Real;

                    Leverancier: Integer;

                  end;

var

  ProductFile  : Product;

  ProductNamen : file of ProductNaam;
```

Het component type van een file mag van elk type zijn behalve van het File type. In het
bovenstaande voorbeeld is file of Product dus niet toegestaan. File variabelen
mogen niet voorkomen in toekenningen of expressies.

13.2 Bewerkingen met files

De volgende hoofdstukken beschrijven de beschikbare procedures voor bewerking van
files. De naam FilVar die hierbij wordt gebruikt is een file variabele die is gedefinieerd
zoals hierboven is beschreven.

13.2.1 Assign

Syntax: Assign (FilVar, Str);

Str is een String expressie waarin een geldige file naam staat. Deze file naam wordt
toegekend aan de file variabele FilVar. Alle verdere bewerkingen op FilVar zullen
worden uitgevoerd naar de file Str. Assign mag nooit gebruikt worden op een reeds
geopende file.

Onder DOS 2 mag men iedere gewenste padnaam invullen.

13.2.2 Rewrite

Syntax: Rewrite (FilVar);

Een nieuwe lege file met de naam zoals die is toegekend aan FilVar wordt aangemaakt en
klaar gemaakt voor bewerking. De file wijzer wordt gezet op het begin van de file ofwel
op component 0. Een reeds bestaande file met dezelfde naam wordt overschreven.

RecordSize is 128 als deze niet wordt opgegven. Het is zaak van de programmeur om er
voor te zorgen dat RecordSize een acceptabele inhoud heeft. RecordSize is alleen op te
geven als FilVar een ongetypeerde file is.

13.2.3 Reset

Syntax: Reset (FilVar);

De file met de naam zoals die is toegekend aan FilVar wordt klaar gemaakt voor
bewerking en de file wijzer wordt aan het begin van de file gezet (naar component 0).
FilVar moet de door Assign toegekende naam bevatten, anders treedt er een looptijdfout
op.

RecordSize is 128 als deze niet wordt opgegeven. Het is zaak van de programmeur om er
voor te zorgen dat RecordSize een acceptabele inhoud heeft. RecordSize is alleen op te
geven als FilVar een ongetypeerde file is.

De grote wijziging die hier gemaakt moest worden is dat de eerste 4 bytes die bij een
getypeerde file worden gebruikt om het aantal records en de record grootte te onthouden
nu overbodig zijn geworden. Het aantal records wordt nu uitgerekend door de file grootte
te delen door de record grootte van de getypeerde file. Verder moet gelden dat de

restwaarde van deze deling nul oplevert. Deze manier van werken is exact hetzelfde als op de PC. Als dit niet overeenkomt dan wordt d.m.v. IOResult een 'Record length mismatch' teruggegeven en GetError geeft dan de waarde 61 terug.

13.2.4 Read

Syntax: Read (FilVar, Varx);

Varx staat voor ÈÈn of meer variabelen van het component type van FilVar gescheiden door komma's. Elke variabele wordt gelezen uit de file en na ieder gelezen component wordt de file wijzer opgehoogd naar het volgende component.

Deze procedure is qua gebruik hetzelfde gebleven.

13.2.5 Write

Syntax: Write (FilVar, Varx);

Varx staat voor ÈÈn of meer variabelen van het component type van FilVar gescheiden door komma's. Elke variabele wordt geschreven naar de file en na ieder gelezen component wordt de file wijzer opgehoogd naar het volgende component.

Deze procedure is qua gebruik hetzelfde gebleven.

13.2.6 Seek

Syntax: Seek (FilVar, N);

Seek verplaatst de file wijzer naar het component N van de file toegekend aan FilVar. N is een Longint expressie. De positie van het eerste component is 0. Merk op dat het mogelijk is om met Seek de file wijzer ÈÈn component voorbij het laatste component van de file te plaatsen. De opdracht:

```
  Seek (FilVar, FileSize (FilVar));
```

Plaatst de file wijzer aan het einde van de file (FileSize geeft het aantal componenten in de file weer, beginnend bij 0. Het terug gegeven aantal is dus ÈÈn groter dan het laatste component).

13.2.7 Flush

Syntax: Flush (FilVar);

Flush maakt de interne buffer van de file FilVar leeg en garandeerd zo dat de buffer is weggeschreven naar de file als er schrijfoperaties hebben plaats gevonden na de laatste file update. Flush zorgt er tevens voor dat de volgende leesoperatie fysiek uit de file wordt gelezen. Flush mag nooit gebruikt worden bij een gesloten file. Bij de opdracht Reset en Close wordt ook een Flush uitgevoerd.

Omdat alle lees- en schrijfopdrachten naar getypeerde en ongetypeerde files direct worden uitgevoerd, er vind dus geen buffering plaats, is de Flush procedure voor deze soorten overbodig.

13.2.8 Close

Syntax: Close (FilVar);

De file geassocieerd met FilVar wordt gesloten en bij een diskfile wordt de directory bij gewerkt. Merk op dat het nodig is om met Close een file te sluiten zelfs als er alleen uit de file is gelezen. Het aantal filehandles zal anders onder DOS 2.xx vlug opraken.

Deze procedure is qua gebruik hetzelfde gebleven.

13.2.9 Erase

Syntax: Erase (FilVar);

De diskfile geassocieerd met FilVar wordt verwijderd. Als de diskfile nog open is, met andere woorden de diskfile is bewerkt met Reset of Rewrite maar nog niet gesloten, dan is het programmeertechnisch gezien beter de diskfile eerst te sluiten met Close voordat hij wordt verwijderd.

Als er een poging wordt gedaan om een device te wissen zal er via de functie GetError de waarde $C1 ('Invalid device operation') teruggegeven worden. Via IOResult wordt de overeenkomstige waarde $20 ('Operation not allowed on a logical device') teruggegeven. Deze procedure is qua gebruik hetzelfde gebleven.

13.2.10 Rename

Syntax: Rename (FilVar, Str);

De diskfile geassocieerd met FilVar krijgt een nieuwe naam welke is opgegeven met de String expressie Str. De disk directory wordt bijgewerkt en toont de nieuwe naam van de file. Alle verdere bewerkingen op FilVar worden uitgevoerd op de file met de nieuwe

naam.

Waarschuwing: Rename mag nooit gebruikt worden bij een open file.

Opmerking: Het is de verantwoordelijkheid van de programmeur om ervoor te zorgen
dat de filenaam Str nog niet bestaat op de schijf. Als de naam wel bestaat kan dit leiden
tot looptijdfout.

De volgende functie heeft als resultaat True als de file naam, die als parameter wordt
meegegeven, bestaat anders geeft hij False terug.

```
type
  Naam = String[66];
  :
  :
function Bestaat (FileNaam: Naam): Boolean;
var
  Fil: File;
begin
  Assign (Fil, FileNaam);
  {$I-}                    {controle uit}
  Reset (Fil);
  {$I+}                    {controle aan}
  Bestaat:=(IOResult=0);
  {
     Bestaat:=(GetError=0)
  }
end;
```

13.3 Standaard file functies

De volgende standaard functies zijn beschikbaar bij files:

13.3.1 Eof

Syntax: EOF (FilVar);

Dit is een booleaanse functie die True teruggeeft als de file wijzer naar de laatste
component van de file wijst. Zoniet dan geeft EOF de waarde False terug.

Deze procedure is qua gebruik hetzelfde gebleven.

13.3.2 FilePos

Syntax: FilePos (FilVar);

Een functie die als waarde de positie van de file wijzer teruggeeft. Dit is altijd een getal
van het type Longint. Het eerste component van de file is 0.

Deze procedure is qua gebruik hetzelfde gebleven.

13.3.3 FileSize

Syntax: FileSize (FilVar);

FileSize is een functie die een Longint getal teruggeeft dat het aantal componenten in de
file weergeeft. Als FileSize de waarde 0 opleverd is de file leeg.

Deze procedure is qua gebruik hetzelfde gebleven.

13.4 Het gebruik van files

Voordat een file gebruikt kan worden moet met de procedure Assign een naam aan de file
worden toegekend. Voordat er lees- of schrijfbewerkingen kunnen worden uitgevoerd
moet de file eerst worden geopend met Rewrite of Reset. Deze Rewrite of Reset zet de file
wijzer naar de eerst component van de file (FilePos(FilVar)=0). Na een Rewrite levert
FileSize altijd 0 op.

Een diskfile kan alleen groter worden door componenten aan het eind van een bestaande
file toe te voegen. De file wijzer kan worden verschoven naar het einde van de file met de
opdracht:

```
Seek(FilVar,FileSize(FilVar));
```

Als het programma klaar is met zijn lees- en schrijfopdrachten naar een file dan moet altijd de procedure Close of Reset worden gebruikt. Als dit niet gebeurd dan kan dit leiden tot het verlies van data omdat de directory niet goed is bijgewerkt.

Het volgende programma creÎert een diskfile genaamd 'product.dat' en schrijft honderd records van het type product naar de diskfile. Dit initialiseerd de diskfile voor willekeurige toegang (records kunnen overal worden gelezen en geschreven in de file).

```
program InitProductFile;

const

  MaxProducten=100;

type

  ProductNaam = String[20];

  Product     = record

                  Naam       : ProductNaam;

                  ItemNummer : Integer;

                  Voorraad   : Real;

                  Leverancier: Integer;

                end;

var

  ProductFile : File of Product;

  ProductRec  : Product;

  I : Integer;

begin

  Assign  (ProductFile, 'PRODUCT.DAT');

  Rewrite (ProductFile);

  with ProductRec do

  begin
```

```
    Naam       :='';
    Voorraad   :=0;
    Leverancier:=0;
    for I:=1 to MaxProducten do
    begin
      ItemNummer:=I;
      Write (ProductFile,ProductRec);
    end;
   end;
  Close (ProductFile);
end.
```

Het volgende programma demonstreert het gebruik van Seek bij willekeurig toegankelijke bestanden. Het programma bewerkt de file die door het vorige programma
is aangemaakt.

```
program BewerkProductFile;
const
  MaxProducten = 100;
type
  ProductNaam  = String[20];
  Product = record
              Naam       : ProductNaam;
              ItemNummer : Integer;
              Voorraad   : Real;
              Leverancier: Integer;
            end;
var
  ProductFile : file of Product;
  ProductRec  : Product;
```

```
  I, Pnr      : Integer;
begin
  Assign (ProductFile, 'PRODUCT.DAT');
  Reset (ProductFile);  ClrScr;
  Write ('Geef product nr: (0=stop) ');
  ReadLn (Pnr);
  while Pnr in [1..MaxProducten] do
  begin
    Seek (ProductFile,Pnr-1);
    Read (ProductFile,ProductRec);
    with ProductRec do
    begin
      Write ('Product naam: (',Naam:20,') ');
      ReadLn (Naam);
      Write ('In Voorraad: (',Voorraad:20:0,') ');
      ReadLn (Voorraad);
      Write ('Leveranciernr.: (',Leverancier:20,') ');
      ReadLn (Leverancier);
      ItemNummer:=Pnr;
    end;
    Seek (ProductFile,Pnr-1);
    Write (ProductFile,ProductRec);
    ClrScr;
    WriteLn;
    Write ('Geef product nummer (0=einde): ');
    ReadLn (Pnr);
  end;
  Close (ProductFile);
end.
```

13.5 Tekst files

In tegenstelling tot alle andere type files zijn tekst files geen opeenvolging van waarden van een bepaald type. Hoewel het grondleggende type van een tekst file het type Char is, is de tekst file gestructureerd in regels. Elke regel wordt afgesloten met een EOL-markering. De EOL-markering is de letterkombinatie CR/LF ofwel de return gevolgd door een regelopvoer (ASCII-waarden 13 en 10). De file is verder afgesloten met een EOF-markering, de CTRL-Z (ASCII-waarde 26). Omdat de lengte van de regels kan variÎren is het niet mogelijk om de beginpositie van een regel te berekenen. Tekst files kunnen daarom alleen sequentieel worden gelezen of geschreven. Daarnaast is het onmogelijk om in dezelfde file zowel te lezen als te schrijven.

13.5.1 Bewerkingen met tekst files

Een tekst file variabele wordt gedefinieerd door achter de variabele naam de standaard type naam Text te plaatsen. De file bewerkingen moeten voorafgegaan worden door een toewijzing met Assign en de file moet geopend zijn met Reset of Rewrite.

Rewrite wordt gebruikt om een nieuwe file te creÎren en hierna is het alleen mogelijk om regels toe te voegen aan de file. Na een Close opdracht wordt automatisch de EOF-markering toegevoegd. Reset wordt gebruikt bij een bestaande file en het is alleen mogelijk om uit de file te lezen. Het is dus niet mogelijk om regels aan een bestaande file toe te voegen.

De karakter in- en uitvoer van en naar tekst files gebeurd met de standaard procedures Read en Write. Hele regels kunnen ineens van of naar de file gelezen of geschreven worden met de bewerkingen ReadLn of WriteLn. De functie Eoln kan gebruikt worden om het einde van een regel te detecteren.

13.5.2 ReadLn

Syntax: ReadLn (FilVar);

Slaat het resterende deel van de regel of de hele regel als de file wijzer aan het begin van

de regel staat over. Met andere woorden, de file wijzer wordt verplaatst tot na de
eerstvolgende EOL-markering.

13.5.3 WriteLn

Syntax: WriteLn (FilVar);

Schrijft een EOL-markering naar de tekst file.

13.5.4 Eoln

Syntax: Eoln (FilVar);

Een booleaanse functie die de waarde True teruggeeft als het einde van de huidige regel is
bereikt. Met andere woorden, de functie is True als de file wijzer zich bevindt op de
positie van de CR van de combinatie CR/LF. Bovendien is de functie True als de file
wijzer op de EOF-markering staat. In alle andere gevallen is de functie False.

13.5.5 SeekEoln

Syntax: SeekEoln (FilVar);

Precies hetzelfde als Eoln, met dit verschil dat er eerst over spaties en tab-posities heen
wordt gesprongen. Het resultaat is ook van het type Boolean.

13.5.6 SeekEof

Syntax: SeekEof (FilVar);

Hetzelfde als SeekEoln, met het verschil dat niet alleen spaties en tab-posities worden
overgeslagen, maar ook EOL-markeringen. Het resultaat is ook van het type Boolean.

Als de functie Eof wordt gebruikt bij tekst files is de teruggegeven waarde True als de file
wijzer wijst naar een EOF-markering (CTRL-Z geeft het einde van de file weer). De
procedures Seek en Flush en de functies FilePos en FileSize kunnen niet worden gebruikt
bij tekst files.

Het volgende voorbeeldprogramma leest een tekst file van de disk en print dit uit via het
voorgedefinieerde apparaat LST.

```
program tekstfiledemo;
var
  FilVar           : Text;
  Regel,ExtraRegel : String[255];
  I                : Integer;
  OnderStreept     : Boolean;
  FileNaam         : String[14];
begin
  OnderStreept := False;
  Write('Geef de filenaam: ');
  ReadLn(FileNaam);
  Assign(FilVar,FileNaam);
  Reset (FilVar);
  while not Eof(FilVar) do
  begin
    ReadLn(FilVar, Regel);
    I:=1;
    ExtraRegel:='';
    for I:=1 to Length(Regel) do
    begin
      if regel[I]<>^S then
      begin
        Write(Lst,regel[I]);
        if OnderStreept then
          ExtraRegel:=ExtraRegel+'_'
        else
          ExtraRegel:=ExtraRegel+' ';
      end
      else
```

```
        OnderStreept:=not OnderStreept;

    end;

    Write(Lst,^M);

    WriteLn(Lst,ExtraRegel);

  end;

end.
```

De procedures Read en Write worden verder behandeld in de hoofdstukken 13.9 tot en met 13.12, inclusief de handige in- en uitvoer mogelijkheden.

13.6 Logische apparaten

In Turbo Pascal worden externe apparaten zoals beeldscherm, printer en modems gezien als logische apparaten, die hetzelfde behandeld worden als files. De volgende logische apparaten zijn beschikbaar:

13.6.1 Con:

Het bedieningspaneel, dus het beeldscherm en het toetsenbord. Uitvoer wordt gestuurd naar het uitvoerapparaat van het besturingssysteem, normaal is dit CRT ofwel het beeldscherm. Invoer komt van het invoerapparaat van het besturingssysteem, normaal het toetsenbord. In tegenstelling tot het TRM apparaat (zie hoofdstuk 13.6.2) voorziet het CON: apparaat in een gebufferde invoer. In het kort komt dit er op neer dat er een tekst file gekoppeld wordt aan het CON: apparaat (invoer van het toetsenbord wordt dus ook gezien als een tekst file!). De Read en ReadLn procedures lezen hiervan een volledige regel in, met mogelijkheid tot bewerken via edit functies. Voor meer details hierover zie hoofdstuk 13.7 en 13.8.

13.6.2 Trm:

Het terminal apparaat. Ook hier wordt het toetsenbord gelezen en het beeldscherm gebruikt voor weergave, tenzij het besturingssysteem hier een ander apparaat aan heeft gekoppeld. Invoer karakters worden weer teruggestuurd naar de uitvoer (het zogenaamde echoÎn), tenzij het controle karakters zijn. Het enige controle karakter dat wel wordt teruggestuurd is de CR (Carriage Return) en deze wordt teruggestuurd als CR/LF.

13.6.3 Kbd:

Het toetsenbord. Alleen invoer is hiervan mogelijk. Invoer wordt verkregen van de console input van het besturingssysteem normaal gesproken het toetsenbord. De invoer wordt niet geÎchoed.

13.6.4 Lst:

Het printer apparaat (alleen uitvoer). Uitvoer wordt verzonden naar het 'list' apparaat van het besturingssysteem. Meestal is dit de printer.

13.6.5 Aux:

Aansturing van hulpapparatuur. Deze uitbreiding wordt toegankelijk als er een serieÎle poort in de computer aanwezig is.

13.6.6 Usr:

Een zelf gedefinieerd apparaat. Uitvoer wordt gestuurd naar een door de programmeur geschreven procedure en invoer komt van een zelf geschreven procedure. Voor meer details over in- en uitvoer zie 17.10 "Definieerbare I/O drivers".

Deze logische apparaten kunnen worden benaderd als voorgedefinieerde files of ze kunnen worden gekoppeld aan file variabelen. In het laatste geval werken ze hetzelfde als diskfiles. Rewrite en Reset werken op files toegekend aan logische apparaten hetzelfde als op files. Close voert geen functie uit en het gebruik van Erase zal leiden tot een I/O-error.

De standaard functies Eof en Eoln werken verschillend bij logische apparaten en diskfiles. Bij een diskfile geeft Eof de waarde True als het volgende karakter in de file een CTRL-Z is of als er fysiek voorbij het einde van de file gelezen wordt. De Eoln functie geeft True als het volgende karakter een CR of een CTRL-Z is. Eof en Eoln kijken dus vooruit in de file.

Omdat het bij een logisch apparaat niet mogelijk is om het volgende karakter te bekijken

werken de functies Eof en Eoln anders bij logische apparaten. De beide functies kijken niet naar het volgende karakter, maar naar het laatst gelezen karakter.

Bij files

Bij logische apparaten

EOLN is True

Volgende karakter is CR, CTRL-Z of EOF is True

Het huidige karakter is CR of CTRL-Z

EOF is True

Volgende karakter is CTRL-Z of het fysieke einde van de file is bereikt.

Huidige karakter is CTRL-Z

Tabel 13 - 1

Een zelfde verschil tussen logische apparaten en diskfiles komt voor bij de ReadLn procedure. Bij diskfiles worden alle karakters gelezen inclusief een CR/LF combinatie.
Bij logische apparaten wordt gelezen tot de eerste CR. Ook hier weer dezelfde reden, namelijk dat het systeem niet vooruit kan kijken bij logische apparaten.

13.7 Standaard files

Er is een alternatief voor het toekennen van tekst files aan logische apparaten zoals hierboven is beschreven. Turbo Pascal biedt een aantal voorgedefinieerde tekst files welke al zijn toegekend aan logische apparaten en voorbereid zijn voor gebruik. Hierdoor hoeft de programmeur de Reset - Rewrite procedure niet meer uit te voeren. Het gebruik van standaard files bespaart bovendien programmacode.

Input

De belangrijkste invoer file. Deze file is toegekend aan het CON: apparaat of aan het TRM: apparaat, zie hieronder voor meer details.

Output

De belangrijkste uitvoer file. Deze file is toegekend aan het CON: apparaat of aan het
TRM: apparaat, zie hieronder voor meer details.

CON:

Toegekend aan het bedieningspaneel. (=toetsenbord en beeldscherm)

TRM:

Toegekend aan het terminal apparaat.

KBD:

Toegekend aan het toetsenbord.

LST:

Toegekend aan het printer apparaat.

AUX:

Toegekend aan hulp apparatuur.

USR:

Toegekend aan zelf gedefinieerde apparatuur.

Merk op dat het gebruik van Assign, Reset, Rewrite en Close op deze file variabelen niet
toegestaan is.

Als de Read procedure wordt gebruikt zonder dat er een file naam is opgegeven, wordt er
altijd een regel ingelezen. De regel moet worden afgesloten met een return. Een
ingetoetste CTRL-Z wordt genegeerd en de ingegeven return wordt niet geÎchoed. Intern

wordt een CTRL-Z aan de regel toegevoegd. Wanneer er dus minder gegevens zijn
gespecificeerd in de ingegeven regel dan worden Char variabelen gevuld met een
CRTL-Z, Strings zijn leeg en numerieke variabelen zijn ongedefinieerd.

De compiler aanwijzing B wordt gebruikt om deze geforceerde Read te beÔnvloeden. De
standaard waarde is {$B+} en in deze stand wordt de Read opdracht zonder een file
variabele altijd gelezen van het bedieningspaneel. Als de {$B-} compiler aanwijzing
geplaatst is aan het begin van het programma (voor het declaratie deel) wordt voor invoer
zonder file naam de standaard input file gebruikt:

Read (v1, v2, ..., vn) is gelijk aan Read (Input, v1, v2, ..., vn)

In dit geval worden regels alleen ingevoerd als de regel buffer is leeg gemaakt. De
compiler aanwijzing {$B-} volgt de definitie van het standaard Pascal, terwijl de
instelling {$B+} niet standaard Pascal is, maar wel een betere controle over invoer
operaties biedt.

Als u niet wilt dat invoer wordt teruggestuurd naar het scherm dient u gebruik te maken
van de standaard file KBD:.

Read (Kbd,Var)

Omdat de standaard files Input en Output vaker worden gebruikt zijn zij gekozen als
standaard file indien geen file naam is opgegeven. De volgende lijst toont de verkorte
tekst file bewerkingen en hun gelijke:

```
   Write(Ch)     Write(Output,Ch)

   Read(Ch)      Read(Input,Ch)

   WriteLn       WriteLn(Output)
```

```
   ReadLn       ReadLn(Input)

   Eof          Eof(Input)

   Eoln         Eoln(Input)
```

Het volgende programma toont het gebruik van de standaard file LST: om de file
ProductFile (zie hoofdstuk 13.4) af te drukken op de printer.

```
program ListProductFile;

const

  MaxProducten = 100;

type

  ProductNaam = String[20];

  Product     = record

                  Naam            : ProductNaam;

                  OnderdeelNummer : Integer;

                  InVoorraad      : Real;

                  Leverancier     : Integer

                end;

var

  ProductFile : file of Product;

  ProductRec  : Product;

  I : Integer;

begin

  Assign(ProductFile,'PRODUCT.DAT');

  Reset(ProductFile);

  for I := 1 to MaxProducten do

  begin

    Read(ProductFile,ProductRec);

    with ProductRec do
```

```
    begin
      if Naam<>'' then
             WriteLn(Lst,'Onderdeel: ',
                OnderdeelNummer:5,' ', Naam:20,
                '  Van: ', Leverancier:5,
                ' Nu in voorraad:', InVoorraad:0:0);
     end;
   end;
   Close(ProductFile)
end.
```

13.8 Tekst in- en uitvoer

In- en uitvoer van data in leesbare vorm gebeurd door middel van tekst files zoals
beschreven in hoofdstuk 13.5.1. Een tekst file kan worden gekoppeld aan een willekeurig
apparaat, dus aan een diskfile of aan ÈÈn van de standaard I/O apparaten. In- en uitvoer op
tekst files wordt gedaan met de standaard procedures Read, ReadLn, Write en WriteLn.
Hierbij wordt een speciale syntax gebruikt voor de parameters waardoor een zeer flexibel
in- en uitvoer systeem ontstaat.

De opgegeven parameters mogen van een verschillend type zijn en de procedures
converteren dit type automatisch naar het type Char. Het Char type wordt gebruikt in
tekst files.

Als de eerste parameter van een I/O procedure de variabele naam is van een tekst file dan
zal de in- of uitvoer plaats vinden naar die file. Is de eerste parameter geen tekst file
variabele, dan zal de in- of uitvoer gebruik maken van de standaard files Input en Output.
Zie hoofdstuk 13.7 voor meer details.

13.9 Read procedure

De Read procedure voorziet in de invoer van karakters, strings en numerieke data. De
schrijfwijze van deze procedure opdracht is:

```
  Read (Var1, Var2, ..., VarN)
```

of

```
  Read (FilVar, Var1, Var2, ..., VarN)
```

Var1, Var2, ..., VarN zijn variabelen van het type Char, String, Byte, Integer, Longint of Real. In het eerste geval worden de variabelen van de standaard file input (normaal het toetsenbord) gelezen. In het tweede geval wordt de invoer gelezen van de tekst file welke daarvoor toegekend is aan FilVar en gereed gemaakt is om uit te lezen.

Bij een variabele van het type Char leest Read ÈÈn karakter uit de file en kent deze toe aan de variabele. Als de file een tekst file is wordt Eoln True als het volgende karakter een CR of een CTRL-Z is. Eof is True als het volgende karakter een CTRL-Z is of het fysieke einde van de file is bereikt. Als de file een logisch apparaat is (inclusief de standaard files Input en Output) is Eoln True, als het gelezen karakter een CR was of als Eof True is. Eof is True als het gelezen karakter CTRL-Z was.

Bij een variabele van het type String leest Read net zoveel karakters als er maximaal in de String passen tenzij een Eoln of een Eof eerst bereikt wordt. Eoln is True als het gelezen karakter een CR was of Eof is True. Eof is True als het laatste gelezen karakter een CTRL-Z was of het fysieke einde van een file is bereikt.

Bij een numerieke variabele (Byte, Integer, Longint of Real) verwacht Read een rij met karakters welke overeenkomt met het formaat van het overeenkomende type. Het formaat staat beschreven in hoofdstuk 3.2. Als er spaties, tabs, CR's of LF's vooraf gaan aan de String dan worden deze overgeslagen. De String mag niet langer zijn dan 30 karakters en moet gevolgd worden door een spatie, tab, CR of een CTRL-Z. Als de String niet overeenkomt met het verwachte formaat zal er een looptijd fout (nr. 10) optreden. Is het formaat wel goed dan zal de karakter rij worden omgezet naar een waarde van het juiste type en toegekend worden aan de variabele. Als er gelezen wordt van een diskfile en de invoer rij is beÎindigd met een spatie of een tab dan zal de volgende Read of ReadLn

beginnen met het karakter dat direct volgt op die spatie of tab. Voor zowel diskfiles als
logische apparaten geldt dat Eoln is True als de String was beÎindigd met CR of een
CTRL-Z. Eof is True als de String was beÎindigd met een CTRL-Z.

Een speciale behandeling van numerieke invoer vindt plaats als Eoln of Eof True is aan
het begin van een Read (b.v. als invoer van het toetsenbord een CR was). In dat geval
wordt er geen nieuwe waarde toegekend aan de variabele en de variabele behoudt zijn
oude waarde.

Als de invoer file is toegekend aan het bedieningspaneel of aan de standaard file input
gelden er speciale regels voor het lezen van variabelen. Bij iedere aanroep van Read of
ReadLn wordt een regel ingelezen via het toetsenbord en opgeslagen in een buffer. Het
lezen van een variabele gebeurd vanuit deze buffer. Deze buffer voorziet tevens in een
mogelijkheid voor regel bewerking tijdens het invoeren.

De volgende bewerkingsmogelijkheden zijn beschikbaar:

Backspace of Del

Gaat ÈÈn karakter terug en verwijderd het karakter dat daar staat. Backspace wordt
ingegeven door de toets BS of door CTRL-H.

Escape of CTRL-X

Gaat terug naar het begin van de regel en verwijderd alle ingevoerde karakters.

CTRL-D

Haalt ÈÈn karakter terug van de vorige ingegeven regel.

CTRL-R

Haalt de vorige ingegeven regel terug.

Return of CTRL-M

BeÎindigd de invoer en plaatst een markering voor het einde van de regel (CR/LF combinatie) in de regel buffer. De CR/LF wordt niet terug gestuurd naar het scherm.

CTRL-Z

Verwijderd de ingegeven regel en plaatst een markering voor het einde van de file (CTRL-Z) in de regel buffer.

De ingegeven regel wordt intern opgeslagen met een CTRL-Z aan het einde toegevoegd. Als er dus minder waarden worden ingetypt als in de parameterlijst staat opgegeven krijgen de niet ingevulde parameters een lege waarde. Karakter variabelen worden op CTRL-Z gezet, Strings blijven leeg en numerieke variabelen blijven ongewijzigd.

Het maximaal aantal karakters dat kan worden ingegeven via het toetsenbord is 126. U kunt dit echter verlagen door een waarde toe te kennen aan de voorgedefinieerde variabele BufLen. Deze waarde van variÎren van 0 t/m 126.

Voorbeeld:

```
  Write ('filenaam (max 14 karakters):');

  BufLen:=14;

  Read(filenaam);
```

Merk op dat toekenningen aan BufLen alleen effect hebben op de eerstvolgende Read. Daarna wordt BufLen terug gezet op 126.

13.10 ReadLn procedure

De ReadLn procedure is gelijk aan de Read procedure met dit verschil dat na het lezen van de laatste parameter de rest van de regel wordt genegeerd. Met andere woorden, alle

karakters tot de volgende CR/LF combinatie (of de volgende CR bij een logisch apparaat)
worden overgeslagen. De schrijfwijze van de procedure opdracht is:

```
  ReadLn (Var1,Var2,...,VarN)
```

of

```
  ReadLn (FilVar,Var1,Var2,...,VarN)
```

Na een ReadLn zal de volgende Read of ReadLn beginnen te lezen aan het begin van de
volgende regel. ReadLn mag ook worden gebruikt zonder parameters:

```
  ReadLn
```

of

```
  ReadLn(FilVar)
```

In dit geval zal de rest van de ingegeven regel worden overgeslagen als er gelezen wordt
van het bedieningspaneel (standaard file input of de file toegekend aan CON:). De
afsluitende CR wordt teruggestuurd als een CR/LF combinatie zoals besproken bij Read
in hoofdstuk 13.9.

13.11 Write procedure

De Write opdracht voorziet in uitvoer van karakters, strings, booleaanse waarden en
numerieke waarden. De schrijfwijze van de procedure opdracht is:

```
  Write (Var1,Var2,...,VarN)
```

of

```
  Write (FilVar,Var1,Var2,...,VarN)
```

Hierbij zijn Var1, Var2,..., VarN (de schrijfparameters) variabelen van het type Char,
String, Boolean, Byte, Integer, Longint of Real. De variabele mag gevolgd worden door
een dubbele punt en een Integer expressie waarin de breedte van het uitvoer veld wordt

gedefinieerd. In het eerste geval wordt de variabele gestuurd naar de standaard uitvoer
file output wat normaal het beeldscherm is. In het tweede geval worden de variabelen
gestuurd naar een tekst file welke is toegekend aan FilVar.

Het formaat van de schrijfparameters hangt af van het type van de variabele. In de
beschrijving die hieronder volgt worden de volgende symbolen gebruikt:

i,m,n staat voor een Integer expressie

l staat voor een Longint expressie

r staat voor een Real expressie

ch staat voor een Char expressie

s staat voor een String expressie

b staat voor een Boolean expressie

13.11.1 Schrijfparameters

ch Het karakter ch wordt uitgevoerd.

ch:n Het karakter ch wordt rechts uitgelijnd uitgevoerd in een veld dat n karakters
breed is. Met andere woorden het karakter wordt voorafgegaan door n - 1 spaties.

s De String wordt uitgevoerd, array's van karakters kunnen ook zo worden
uitgevoerd omdat zij compatibel zijn met Strings.

s:n De String wordt rechts uitgelijnd uitgevoerd in een veld dat n karakters breed
is. Met andere woorden s wordt voorafgegaan door n - Length(s) spaties.

b Afhankelijk van de waarde van b wordt het woord True of het woord False
uitgevoerd.

b:n Afhankelijk van de waarde van b wordt het woord True of het woord False
rechts uitgelijnd uitgevoerd. In een veld dat n karakters breed is.

i De decimale weergave van de waarde van i wordt uitgevoerd.

i:n De decimale weergave van de waarde van i wordt rechts uitgelijnd uitgevoerd
in een veld dat n karakters breed is.

l De decimale weergave van de waarde van l wordt uitgevoerd.

l:n De decimale weergave van de waarde van l wordt rechts uitgelijnd uitgevoerd
in een veld dat n karakters breed is.

r De decimale weergave van de waarde van r wordt uitgevoerd in een veld van
18 karakters breed. Hierbij wordt gebruikt gemaakt van het floating point formaat.

Voor r>=0.0 is het formaat ' #.##########E*##'

Voor r< 0.0 is het formaat ' -#.##########E*##'

Hierbij staan ' ' en ' 'voor een spatie, # staat voor een cijfer en * staat voor
een plus of een min teken.

r:n De decimale weergave van de waarde van r wordt rechts uitgelijnd in een veld
van n karakters breed. Hierbij wordt gebruikt gemaakt van het floating point
formaat.

Voor r>=0.0 is het formaat <spaties>#.cijfersE*##

Voor r< 0.0 is het formaat <spaties>-#.cijfersE*##

Hierbij staat '<spaties>' voor 0 of meer spaties, cijfers staat voor 1 tot 10
cijfers, # staat voor een cijfer en * staat voor een plus of een min teken. Als er
na een decimale punt minimaal 1 karakter wordt uitgevoerd is de minimale
waarde voor n 7 en 8 als r kleiner is dan als 0.0.

r:n:m De decimale weergave van de waarde van r wordt rechts uitgelijnd weergegeven in een veld dat n karakters breed is en m karakters na de punt heeft
staan. Als m 0 is wordt er geen punt en geen decimaal deel weergegeven. Als
m groter is dan 24 wordt het normale floating point formaat gebruikt. Het
uitgevoerde getal wordt voorafgegaan door zoveel spaties dat het veld n
karakters breed wordt.

13.12 WriteLn procedure

De WriteLn procedure is gelijk aan de Write procedure met als verschil dat een CR/LF
combinatie wordt uitgevoerd na de laatste waarde. De schrijfwijze van de WriteLn
opdracht is:

```
WriteLn(Var1,Var2,...,VarN)
```

of

```
WriteLn(FilVar,Var1,Var2,...,VarN)
```

Een WriteLn zonder schrijf parameters voert een lege regel uit, bestaande uit een CR/LF combinatie:

```
WriteLn
```

of

```
WriteLn(FilVar)
```

13.13 Niet getypeerde files

Niet getypeerde files zijn in- en uitvoer kanalen die op een erg laag niveau werken. De diskfile wordt standaard benaderd met een record formaat van 128 bytes, tenzij er een andere waarde bij het openen van de file wordt meegeven, Reset en ReWrite.

In- en uitvoer naar niet getypeerde files vindt plaats zonder gebruik te maken van een interne buffer. Het data transport vindt plaats tussen de disk en de variabele. Omdat er geen interne buffer nodig is, gebruiken niet getypeerde files minder geheugen dan andere file variabelen. Omdat een niet getypeerde file compatibel is met alle andere file typen is het daarom handig dit type te gebruiken als de file moet worden gewist (Erase) of een andere naam moet krijgen (Rename) of andere niet in- of uitvoer bewerkingen moet ondergaan.

Een niet getypeerde file wordt gedeclareerd met het gereserveerde woord file:

```
var
  DataFile : file;
```

13.13.1 BlockRead / BlockWrite

Alle bewerkingen die toegestaan zijn op standaard files zijn ook van toepassing op niet getypeerde files met uitzondering van Read, Write en Flush. Read en Write zijn

vervangen door twee bewerkingen met een veel hogere overdrachtsnelheid: BlockRead
en BlockWrite. De schrijfwijze van deze twee procedures is:

```
  BlockRead(FilVar,Var,Recs)

  BlockWrite(FilVar,Var,Recs)
```

of

```
  BlockRead(FilVar,Var,Recs,Result)

  BlockWrite(FilVar,Var,Recs,Result)
```

Hierin is FilVar een variabele naam van een niet getypeerde file en Var is een variabele
van een willekeurig type. Recs is een Integer expressie waarmee het aantal records wordt
opgegeven die worden verplaatst tussen de file FilVar en de variabele Var. De optionele
parameter Result geeft aan hoeveel records er daadwerkelijk zijn verplaatst. De grootte
van een record is standaard 128 bytes, tenzij anders opgegeven.

De data overdracht begint vanaf het eerste byte dat gebruikt wordt door de variabele Var.
De programmeur moet er voor zorgen dat de variabele Var groot genoeg is om de
volledige data overdracht te kunnen bevatten. Een aanroep van BlockRead of BlockWrite
zorgt er tevens voor dat de file wijzer Recs wordt aangepast.

Voordat een file kan worden bewerkt met BlockRead of BlockWrite moet de file eerst
worden voorbewerkt met Assign en Rewrite of Reset. Rewrite opent een nieuwe file en
Reset opent een bestaande file. Na bewerking moet de file met Close worden afgesloten.

De standaard functie Eof werkt hetzelfde als bij getypeerde files. Ook de functies FilePos
en FileSize en de standaard procedure Seek werken hetzelfde, gebruik makend van de
eventueel opgegeven recordgrootte.

Het volgende programma gebruikt niet getypeerde files om een willekeurige file te

copiÎren. Merk op dat de optionele vierde parameter in BlockRead wordt gebruikt om te
kijken hoeveel records er daadwerkelijk zijn gelezen van de source file.

```
program FileCopy;

const

  RecSize = 1;

  BufSize = 25600;

var

  Source, Dest : File;

  SourceNaam,

  DestNaam: string[63];

  Buffer  : array[1..BufSize,1..RecSize] of Byte;

  RecsRead: Integer;


begin

  Write('Naam originele file: ');

  ReadLn(SourceNaam);

  Assign(Source,SourceNaam);

  Reset(Source, RecSize);

  Write('Schrijven naar: ');

  ReadLn(DestNaam);

  Assign(Dest,DestNaam);

  Rewrite(Dest, RecSize);

  repeat

    BlockRead(Source,Buffer,BufSize,RecsRead);

    BlockWrite(Dest,Buffer,RecsRead);

  until RecsRead =0;

  Close(Source);

  Close(Dest)
```

end.

13.14 I/O controle

De compiler aanwijzing I wordt gebruikt om de controle op in- en uitvoer aan of uit te zetten. De ingestelde waarde is actief {$I+} wat er voor zorgt dat er een looptijdfout optreed als er een fout ontstaat tijdens het in- of uitvoeren van data. Als er een fout optreed zal het programma worden afgebroken en wordt de foutmelding op het scherm geplaatst.

Als de controle uit staat {$I-}, dan wordt er wel gecontroleerd, maar wordt het programma niet afgebroken. Het is echter niet mogelijk nog meer I/O instructies uit te voeren totdat de functie IOResult is aangeroepen. Als IOResult wordt aangeroepen wordt de foutconditie gewist en kan er opnieuw I/O plaatsvinden. Het is nu de taak van de programmeur om de juiste maatregelen te treffen bij de desbetreffende foutmelding. Als de functie IOResult een nul teruggeeft is de in- of uitvoer operatie geslaagd. In appendix G staat een overzicht van de foutmeldingen en het bijbehorende nummer.

Merk op dat de foutconditie wordt gewist bij het opvragen van IOResult. Als IOResult meer dan ÈÈn keer wordt aangeroepen zullen alle andere keren dus de waarde nul teruggeven totdat er weer een fout optreed.

Waarschuwing: De functie GetError geeft een gedetailleerdere melding dan IOResult.
Het wordt dus sterk aangeraden om bij foutcontroles, die te maken hebben met file bewerkingen, de GetError functie te gebruiken. Voor een complete lijst met foutmeldingen die de GetError kan geven verwijzen we door naar de DOS 2 documentatie. De GetError geeft zowel onder DOS 1 als onder DOS 2 dezelfde waarden.
Het aanroepen van de GetError functie heeft tot gevolg dat IOResult op nul wordt gezet.

In het volgende voorbeeld wordt iedere keer opnieuw een file naam gevraagd totdat de ingegeven file naam voorkomt op de disk:

```
program OpenFileIn;
```

```
var OK : Boolean;
    FileIn:     file;
    FileInNaam: string[63];
    DosError:  integer;
begin
  DosErrorHandlerOn;

  repeat
    Write('Geef de naam van de invoerfile: ');
    ReadLn(FileInNaam);
    Assign(FileIn,FileInNaam);
    {$I-} Reset(FileIn); {$I+}
    OK:=(IOResult=0);
    if not OK then
      WriteLn('Kan file ',FileInNaam,' niet vinden.');

    Assign(FileIn,FileInNaam);
    {$I-}
    Reset(FileIn);
    DosError:=GetError;
    {$I+}
    if DosError <> 0 then
    begin
      WriteLn('Kan file ',FileInNaam,' niet vinden.');
      WriteLn('GetError: ', DosError);
    end;
  until OK

  DosErrorHandlerOff;
```

end.

Als de compiler aanwijzing I niet actief is {$I-} dan moeten de volgende standaard procedures worden gecontroleerd met de functie IOResult:

Assign	Close	Read	Rewrite
BlockRead	Erase	ReadLn	Seek
BlockWrite	Execute	Rename	Write
Chain	Flush	Reset	WriteLn

De procedures DosErrorHandlerOn en DosErrorHandlerOff zorgen er voor dat d.m.v. de GetError functie ook fouten als 'Disk not ready' kunnen worden opgevraagd.

14. WIJZER TYPE (pointer)

De variabelen die tot nu toe zijn besproken zijn allemaal statisch. De vorm en het formaat zijn allemaal voorgedefinieerd en bestaan allemaal gedurende het hele blok waarin ze zijn gedeclareerd. Programma's kunnen echter behoefte hebben aan data structuren die variÎren in vorm en formaat tijdens uitvoering van het programma. Voor dit doel worden dynamische variabelen gebruikt die worden aangemaakt als ze nodig zijn en kunnen nadien gewist worden.

Deze dynamische variabelen worden niet op dezelfde manier gedeclareerd als statische variabelen. Ze kunnen dan ook niet direct benaderd worden via namen. In plaats hiervan wordt er een speciale variabele gebruikt die het geheugen adres van de gewenste variabele aanwijst. Deze speciale variabele wordt een wijzer genoemd (Engl.: Pointer).

14.1 Definitie van wijzer variabelen

Het wijzer type wordt gedefinieerd met het symbool ^ gevolgd door de type naam van de dynamische variabele. De dynamische variabele kan worden benaderd met behulp van de wijzer variabele van dit type.

Het volgende programma toont de declaratie van een record met zijn aanhangende wijzer.
Het type PersoonPointer is gedeclareerd als een wijzer naar variabele van het type
PersoonRecord:

```
type

  PersoonPointer = ^PersoonRecord;

  PersoonRecord  = record

                     Naam     : String[50];

                     Werk     : String[50];

                     Volgende : PersoonPointer

                   end;

var

  EerstePersoon,

  LaatstePersoon,

  NieuwePersoon : PersoonPointer;
```

De variabelen EerstePersoon, LaatstePersoon en NieuwePersoon zijn nu wijzer variabelen die kunnen wijzen naar records van het type PersoonRecord. Zoals
hierboven is te zien kan de type naam in een wijzer type definitie wijzen naar een naam
die nog niet is gedefinieerd, dit is alleen bij wijzers mogelijk.

14.2 Uitbreiden van variabelen (New)

Voordat het enig zin heeft om deze wijzer variabele te gebruiken moeten we natuurlijk
eerst een variabele hebben waar we naar kunnen wijzen. Nieuwe variabelen van een
willekeurig type worden aangemaakt met de standaard procedure New. De procedure
heeft ÈÈn parameter welke een wijzer moet zijn naar een variabele van het type dat we
willen creÎren. Een nieuwe variabele van het type PersoonRecord kan worden
gecreÎerd met de opdracht:

```
  New (EerstePersoon)
```

Dit heeft het effect dat EerstePersoon wijst naar een dynamisch aangemaakt record van het type PersoonRecord.

Toewijzingen tussen twee wijzer variabelen zijn toegestaan zolang beide wijzers van hetzelfde type zijn. Wijzers van hetzelfde type kunnen ook worden vergeleken met de relationele operatoren = en <>. Deze vergelijking levert een Boolean resultaat op (True of False).

De wijzer waarde nil is compatibel met alle wijzer typen. Nil wijst naar geen ÈÈn dynamische variabele en mag worden toegekend aan een wijzer variabele om aan te geven dat er geen bruikbare wijzer aanwezig is. Nil kan ook worden gebruikt in vergelijkingen.

Variabelen aangemaakt met de standaard bewerking New worden opgeslagen in een stapel structuur (Engl.: Stack) die de heap wordt genoemd. Het Turbo Pascal systeem controleert de heap door het bijhouden van een heap wijzer die in het begin van een programma wordt geinitialiseerd op het adres van het eerste vrije byte in het geheugen.
Bij iedere aanroep van New wordt de heap wijzer verschoven naar de top van het vrije geheugen. De verschuiving is even groot als het overeenkomende formaat van de nieuwe dynamische variabele.

14.3 Mark en Release

Als de dynamische variabele niet langer nodig is voor het programma worden de standaard bewerkingen Mark en Release gebruikt om het geheugen vrij te geven dat door de variabele werd gebruikt. De Mark bewerking koppelt de waarde van de heap wijzer aan een variabele. De schrijfwijze van de Mark bewerking is:

```
  Mark (PVar);
```

Hierbij is PVar een variabele van het wijzer type. De Release bewerking zet de heap
wijzer naar het adres dat aangegeven is in zijn parameter. De schrijfwijze is:

```
  Release (PVar);
```

Hierbij is PVar een variabele van het wijzer type, voorheen aangewezen door Mark.
Release verwijderd dus alle dynamische variabelen boven dit adres en kan geen ruimte
vrij geven die gebruikt wordt door variabelen in het midden van de heap. Als u dit wilt
doen moet u gebruik maken van de bewerking Dispose (zie hoofdstuk 14.5) in plaats van
Mark en Release.

De standaard bewerking MemAvail is aanwezig om vast te stellen hoeveel vrij heap
geheugen er nog beschikbaar is.

14.4 Het gebruik van wijzers

Stel dat we de bewerking New hebben gebruikt om een serie records van het type
PersoonRecord te creÎren (zoals in het voorbeeld in hoofdstuk 3). Het veld Volgende in
elk record wijst naar het volgende PersoonRecord dat is gecreÎerd. Het volgende
voorbeeld zal door de lijst heen lopen en de inhoud van elk record afdrukken
(EerstePersoon wijst naar de eerste persoon in de lijst):

```
while EerstePersoon <> nil do

  with EerstePersoon^ do

  begin

    WriteLn (naam,' is een ', Werk);

    EerstePersoon:=Volgende;

  end;
```

Het volgende programma demonstreert het gebruik van wijzers om een lijst bij te houden

met namen en beroepen. Namen en beroepen zullen worden ingelezen totdat er een
blanco naam wordt ingevoerd. Daarna wordt de volledige lijst afgedrukt. Tot slot wordt
het gebruikte geheugen vrij gegeven. De wijzer variabele HeapTop wordt alleen gebruikt
om de begin waarde van de heap wijzer in op te slaan. Deze is gedefinieerd als
^Integer (wijzer naar een Integer).

```
program Jobs;

type

  PersoonRecord = record

                    Naam     : String[50];

                    Werk     : String[50];

                    Volgende : PersoonPointer

                  end;

     PersoonPointer = ^PersoonRecord;

var

  EerstePersoon,

  LaatstePersoon,

  NieuwePersoon : PersoonPointer;

  Naam          : String[50];

  HeapTop       : Integer;

begin

  EerstePersoon:=nil;

  Mark(HeapTop);

  repeat

    Write('Voer naam in : ');

    ReadLn(Naam);

    if Naam <> '' then

    begin

      New(NieuwePersoon);
```

```
      NieuwePersoon^.Naam:=Naam;

      Write('Voer beroep in: ');

      ReadLn(NieuwePersoon^.Werk);

      WriteLn;

      if EerstePersoon = nil then

        EerstePersoon := NieuwePersoon

      else LaatstePersoon^.Volgende := nil;

    end;

  until Naam='';

  WriteLn;

  while EerstePersoon <> nil do

    with EerstePersoon^ do

    begin

      WriteLn(Naam,' is een ', Werk);

      EerstePersoon:=Volgende

    end;

  Release(HeapTop)

end.
```

14.4.1 Dispose

Syntax: Dispose (PVar: Pointer);

In plaats van Mark en Release kan ook de standaard bewerking Dispose worden gebruikt
voor het terug krijgen van ruimte op de heap.

Merk op dat Dispose en Mark / Release een volledig andere aanpak van het geheugen
beheer gebruiken, deze twee gaan nooit samen. Een programma moet dus kiezen voor de
Dispose of de Mark / Release om het geheugen van de heap vrij te maken. Het door
elkaar gebruiken zal onvoorspelbare resultaten opleveren.

De schrijfwijzen van Dispose is:

```
  Dispose (PVar);
```

Hierbij is PVar een wijzer variabele.

Met Dispose is het mogelijk om het geheugen dat gebruikt wordt door ÈÈn speciale wijzer variabele vrij te geven in tegenstelling tot Mark en Release welke het gehele geheugen vanaf een bepaalde wijzer en hoger vrij geeft.

In de volgende tabel staan een aantal variabelen weergeven op de heap. Er wordt getoond wat het resultaat van de opdracht Dispose(Var3) en Mark(Var3) / Release(Var3).

Heap

na Dispose

na Mark / Release

Var1

Var1

Var1

Var2

Var2

Var2

Var3

Var4

Var4

Var5

Var5

Var6

Var6

Var7

Var7

Tabel 14 - 1

Na het verwijderen van een wijzer variabele met Dispose zullen er dus gaten met vrije

geheugen ruimte in de heap ontstaan. Deze ruimte wordt door het gebruikt van New weer opgevuld als de nieuwe wijzer variabele in zo'n gat past.

14.4.2 GetMem

Syntax: GetMem (PVar: Pointer; Aantal: Integer);

De standaard procedure GetMem wordt gebruikt om ruimte op de heap aan te vragen. In tegenstelling tot New, welke zoveel ruimte aanvraagt als nodig is voor het wijzer type. Bij GetMem controleert de programmeur de hoeveelheid ruimte die wordt aangevraagd.
GetMem wordt gebruikt met twee parameters:

```
  GetMem (PVar,Aantal);
```

Hierbij is PVar een variabele van een wijzer type en Aantal een Integer expressie die aangeeft hoeveel bytes er moeten worden aangevraagd.

14.4.3 FreeMem

Syntax: FreeMem (PVar: Pointer; Aantal: Integer);

De standaard procedure FreeMem wordt gebruikt om een volledig blok geheugen vrij te maken van de heap. Het is dus de tegenhanger van GetMem. FreeMem wordt aangeroepen met twee parameters:

```
  FreeMem (PVar,Aantal);
```

Hierbij is PVar een variabele van een wijzer type en Aantal een Integer expressie die aangeeft hoeveel bytes er moeten worden vrij gegeven. Aantal moet precies even groot zijn als de hoeveelheid geheugen die was aangevraagd door GetMem.

14.4.4 MaxAvail

Syntax: MaxAvail;

De standaard functie MaxAvail geeft het formaat terug van het grootste vrije geheugenblok op de heap. Het resultaat is van het type Integer en als er meer dan 32767 bytes beschikbaar zijn geeft MaxAvail een negatief getal terug. Het correcte getal kan

worden berekend door 65536(.0)+MaxAvail toe te kennen aan een Real of een Longint.
Als de waarde aan een real moet worden toegekend is de constante van het type Real.

14.5 Hints

Merk op dat er geen begrenzingscontrole wordt uitgevoerd op wijzers. Het is de verantwoordelijkheid van de programmeur dat een wijzer naar een toegestaan adres wijst.

15. PROCEDURES EN FUNCTIES

Een Pascal programma bestaat uit ÈÈn of meer blokken, die weer in blokken verdeeld kunnen zijn, enzovoort. Zo'n blok kan een procedure zijn of een functie (algemeen subprogramma genoemd). Een procedure is dus een zelfstandig deel in het programma en kan door een procedure aanroep vanuit een willekeurige plaats in het programma worden aangeroepen (zie hoofdstuk 6.1.2). Een functie is eigenlijk hetzelfde, maar deze berekend ook een waarde, mits de naam van de functie binnen de functie wordt gevuld (zie hoofdstuk 5.2) en geeft deze dan terug.

15.1 Parameters

Waarden kunnen aan procedures en functies met behulp van parameters worden doorgegeven. Dit geeft de mogelijkheid een subprogramma met verschillende waarden uit te laten voeren en daardoor ook verschillende resultaten te krijgen.

De procedureaanroep en de functietoewijzing, die het subprogramma aanroept, kan een parameterlijst bevatten, dit zijn de actuele parameters. Deze worden aan de formele parameters overgegeven, welke in de kop van het subprogramma bepaald zijn. De volgorde van de overdracht moet overeen stemmen met de volgorde van deze definitie.
Pascal ondersteund twee verschillende methoden van overdracht: overdracht van waarden en overdracht van referenties. Bij een overdracht d.m.v. een referentie verandert de formele parameter. Het verschil tussen een waarde overdracht en een referentie overdracht is: dat een waarde parameter niet kan veranderen vanuit een functie en een referentie parameter wel.

Als een parameter door zijn waarde overgedragen wordt, is de formele parameter een lokale variabele in het subprogramma en veranderingen van de formele parameter heeft geen invloed op de actuele parameter. De actuele parameter kan elke willekeurige expressie zijn, inclusief een variabele van het zelfde type als de formele parameter. Zulke parameters heten waarde-parameters en deze worden zoals in het onderstaande voorbeeld in het subprogrammahoofd gedeclareerd. Dit voorbeeld en de volgende voorbeelden laten procedure-prototypen, functie-prototypen en de verschillen hiervan zien.

```
procedure Voorbld(Num1, Num2: Nummer; Str1, Str2: Txt);
```

Nummer en Txt zijn vooraf gedefinieerde typen (bijvoorbeeld Integer of String[255]) en Num1, Num2, Str1, Str2 zijn formele parameters, die de waarden van de actuele parameters overnemen. De typen van de formele en de actuele parameters moeten hetzelfde zijn.

Zorg ervoor dat het type van de parameters in het parameterdeel, alleen een voorgedefinieerd type mag zijn. Dus er mag alleen een type naam staan. Daarom is het volgende:

```
procedure Keuze(Model: array[1..500] of Integer);
```

niet toegestaan.

In plaats daarvan zou het gewenste type in de type definitie van het desbetreffende blok gedefinieerd moeten worden en de type naam zou dan in het prototype gebruikt moeten worden:

```
type Bereik = array[1..500] of Integer;

procedure Select(Model: Bereik);
```

Als een parameter door referentie overgedragen wordt, is de formele parameter dezelfde als de actuele parameter tijdens de uitvoering van het subprogramma. Elke verandering van de formele parameter geldt ook voor de actuele parameter, die daarom ook een variabele moet zijn en geen constante. Parameters die door referentie overgedragen worden, worden variabele parameters genoemd en worden als volgt gedeclareerd:

```
procedure Voorbld(var Num1, Num2: Nummer);
```

Waarde parameters en variabele parameters kunnen in dezelfde procedure gemengd worden, zoals in het volgende voorbeeld:

```
procedure Demo(var Num1, Num2: Nummer; Str1, Str2: Txt);
```

waarin Num1 en Num2 variabele parameters zijn en Str1 en Str2 waarde parameters.

Alle adresberekeningen worden op het tijdstip van de procedureaanroep gedaan. Als een variabele een element uit een array is, wordt de index gecontroleerd voordat het subprogramma aangeroepen wordt.

Let op, dat bestandparameters altijd als variabele parameter gedeclareerd moeten worden.

Als een grote data structuur, zoals bijvoorbeeld een array, aan een subprogramma als een parameter overgedragen moet worden, spaart het gebruik van variabele parameters tijd en geheugen, omdat dan alleen het adres van de actuele parameter aan het subprogramma overgedragen wordt. Een waarde parameter zou tijd en geheugen voor het aanmaken van een extra copie van de data structuur nodig hebben.

15.1.1 Verlichten van de parameter-type controle

Normaal gesproken moet bij het gebruik van een variabele parameter de actuele
parameter gelijk zijn aan de formele parameter. Subprogramma's die variabele
parameters van het type String gebruiken, functioneren alleen met Strings van exact
dezelfde lengte, zoals het in het hoofdprogramma gedefinieerd is. Deze beperking kan
door de V compiler aanwijzing opgeheven worden. De vooringestelde actieve status
{$V+} betekend een strenge type controle, terwijl de passieve status {$V-} de type
controle verlicht en het toestaat actuele parameters met elke String lengte over te dragen,
ongeacht de lengte van de formele parameter.

Voorbeeld:

```
program Decodeer;

{$V-}

type

  WerkString = String[255];

var

  Line1: String[80];

  Line2: String[100];



procedure Decode(var Regel: WerkString);

var I : Integer;

begin

  for I:=1 to Length(Regel) do

    Regel[I]:=Chr(Ord(Regel[I])-30);

end;



begin

  Line1:='Dit is een geheime boodschap';

  Encode(Line1);

  Line2:='Hier is een tweede (langere) boodschap';
```

```
  Encode(Line2);

end.
```

15.1.2 Ongetypeerde variabele parameters

Als het type van de formele parameter niet gedefinieerd is, dan betekend dit dat de type
definitie in het parameterdeel van het subprogramma-hoofd ontbreekt, dan wordt deze
parameter ongetypeerd genoemd. Daardoor kan de actuele parameter van elk willekeurig
type zijn.

De ongetypeerde formele parameter zelf is niet compatibel met de andere data typen en
kan daarom alleen dan gebruikt worden als het data type geen rol speelt. Bijvoorbeeld als
parameter voor Addr, BlockRead, BlockWrite, FillChar, Move of voor de adresspecificatie van een absolute variabele.

De WisselVar procedure in het volgende voorbeeld demonstreert het gebruik van
ongetypeerde parameters. De waarden van A1 en A2 worden omgewisseld.

```
procedure WisselVar(var A1p, A2p; Maat: Integer);

type

  A = array[1..MaxInt] of Byte;

var

  A1    : A absolute A1p;

  A2    : A absolute A2p;

  Tijd  : Byte;

  Teller: Integer;

begin

  for Teller:=1 to Maat do

  begin

    Tijd:=A1[Teller];

    A1[Teller]:=A2[Teller];

    A2[Teller]:=Tijd;

  end;
```

```
end;
```

Aangenomen dat de volgende definities zijn opgenomen:

```
type
  Matrix = array[1..50,1..25] of Real;
var
  TestMatrix, BestMatrix : Matrix;
```

dan kan men WisselVar gebruiken om de waarden van de twee matrices te verwisselen:

```
  WisselVar(TestMatrix, BestMatrix, SizeOf(Matrix));
```

15.2 Procedures

Een procedure kan vooraf gedeclareerd zijn of door de programmeur zelf gedeclareerd worden. Voor gedeclareerde procedures zijn delen van het Turbo Pascal systeem (runtime library) en kunnen zonder verdere aanduidingen aangeroepen worden.

Een door de gebruiker vastgestelde procedure kan de naam van een standaard procedure hebben, maar dan wordt deze standaard procedure onbruikbaar binnen het blok waarin deze zelfgedefinieerde procedure actief is.

15.2.1 Procedure declaratie

De procedure declaratie bestaat uit een procedure hoofd gevolgd door een blok, die uit een declaratie deel en een opdrachtdeel bestaat.

Het procedure hoofd bestaat uit het gereserveerde woord procedure gevolgd door de naam van de procedure, eventueel gevolgd door een formele parameterlijst, zoals in hoofdstuk 15.1 beschreven is.

Voorbeelden:

```
  procedure LogIn;

  procedure Positie(X,Y: Integer);

  procedure Bereken(var Data: Matrix; Schaal: Real);
```

Het declaratie deel van een procedure ziet er hetzelfde uit als dat van een programma.
Alle namen die gedeclareerd zijn in de formele parameterlijst en in het declaratie deel zijn
lokaal ten opzichte van deze procedure en ten opzichte van een nieuwe procedure binnen
deze procedure. Dit wordt het bereik van de naam genoemd. Een procedure mag gebruik
maken van elke willekeurige variabele of constante die buiten zijn procedure is
gedefinieerd, maar wel in hetzelfde blok valt.

Het deel met de opdrachten specificeert de commando's die uitgevoerd moeten worden en
is op dezelfde manier opgebouwd als een samengestelde opdracht (zie hoofdstuk 6.2.1).
Als de naam van een procedure binnen zichzelf wordt gebruikt zal de procedure zichzelf
aanroepen en zo een recursie veroorzaken. Merk op dat voor het goed werken van deze
recursie de compiler geen absolute code moet genereren. Hiervoor moet de compiler
aanwijzing {$A-} moet worden gebruikt. Standaard staat de compiler aanwijzing
{$A+} ingesteld, (zie appendix C).

Het volgende voorbeeld toont een programma dat gebruik maakt een procedure en ook
parameters doorgeeft aan die procedure. Omdat de actuele parameter in dit voorbeeld
iedere keer (kleine) expressies en constanten zijn, moet de formele parameter een waarde
parameter zijn.

```
program RechtHoek;

var

  I: Integer;

procedure TekenVlak(X1, Y1, X2, Y2: Integer);

var I : Integer;

begin
```

```
  GotoXY(X1, Y1);
  for I:=X1 to X2 do
    Write('-');

  for I:=Y1+1 to Y2 do
  begin
    GotoXY(X1, I); Write('!');
    GotoXY(X2, I); Write('!');
  end;

  GotoXY(X1, Y2);
  for I:=X1 to X2 do
    Write('-');
end;        {einde procedure TekenVlak}

begin
  ClrScr;

  for I:=1 to 5 do
    TekenVlak(I*4, I*2, 10*I, 4*I);

  TekenVlak(1, 1, 80, 23);
end.
```

Als de formele parameters in een procedure veranderen is het vaak nodig dat de aanroepende variabelen ook mee veranderen. In dit soort gevallen wordt voor de declaratie van een variabele het gereserveerde woord var opgenomen.

De volgende procedure is daar een voorbeeld van:

```
procedure Wissel(var A, B: Integer);
var Tijd : Integer;
begin
  Tijd:=A; A:=B; B:=Tijd
end;
```

Als dit wordt aangeroepen met de opdracht:

```
  Wissel(I,J);
```

dan zullen de waarden van I en J worden omgedraaid.

Als de definitie van de procedure als volgt was geweest:

```
  procedure Wissel(A, B: Integer);
```

Dan zouden de waarden van I en J niet zijn veranderd.

15.2.2 Standaard procedures

Turbo Pascal kent een aantal standaard procedures. Deze zijn:

1) String bewerkingsprocedures (beschreven in hoofdstuk 8.5)

2) File bewerkingsprocedures (beschreven in hoofdstuk 13.2 / 13.5 / 13.13.1)

3) Procedures voor de toewijzing van dynamische variabelen (beschreven in hoofdstuk 14.3 / 14.4)

4) In- en uitvoer procedures (beschreven in hoofdstuk 13.5)

Daarnaast bestaan nog de volgende standaard procedures:

15.2.2.1 ClrEol

Syntax: ClrEol

Wist alle tekens vanaf de cursor positie tot het einde van de regel, zonder de cursor te
bewegen.

15.2.2.2 ClrScr

Syntax: ClrScr

Wist het beeldscherm en zet de cursor in de linkerbovenhoek. Let erop dat bij veel
schermen de standaard instellingen teruggezet worden als het beeldscherm gewist wordt.

De door de gebruiker ingestelde eigenschappen kunnen hierdoor dus veranderd zijn.

15.2.2.3 CrtInit

Syntax: CrtInit

Stuurt de Terminal Initialisatie String, die in de installatie procedure gedefinieerd is naar
het scherm. Dit wordt niet meer ondersteund vanaf versie 3.3.

15.2.2.4 CrtExit

Syntax: CrtExit

Stuurt de Terminal Reset String, die in de installatie procedure gedefinieerd is naar het
scherm. Dit wordt niet meer ondersteund vanaf versie 3.3.

15.2.2.5 Delay

Syntax: Delay(Time)

De procedure Delay zorgt voor een pause, die ongeveer zoveel milliseconden duurt, als
de parameter Time aangeeft (dit moet een heel getal zijn). De exacte tijd kan op
verschillende systemen verschillend zijn.

15.2.2.6 DelLine

Syntax: DelLine

Wist de regel waarin de cursor staat en beweegt alle regels daaronder naar boven.

15.2.2.7 InsLine

Syntax: InsLine

Voegt een lege regel tussen op de cusor positie. Alle regels daaronder worden een regel
naar onderen bewogen en de onderste regel verdwijnt van het scherm.

15.2.2.8 GotoXY

Syntax: GotoXY(Xpos, Ypos)

Beweegt de cursor naar de positie die door de Integer expressies Xpos (kolom) en Ypos (regel) aangegeven wordt. De linkerbovenhoek is de positie (1,1).

15.2.2.9 Exit

Syntax: Exit

Verlaat het huidige blok. Als Exit in een subprogramma uitgevoerd wordt, zorgt dit voor het verlaten van het subprogramma. Als Exit in het hoofdprogramma uitgevoerd wordt, veroorzaakt dit het afbreken van het programma. Een aanroep van Exit kan vergeleken worden met een goto opdracht die naar een adres wijst vlak voor het einde van het huidige blok.

15.2.2.10 Halt

Syntax: Halt

Onderbreekt het programma en keert terug naar het besturingssysteem.

15.2.2.11 LowVideo

Syntax: LowVideo

Ingebouwd voor systemen die een video aansturing gebruiken met attributen. Hiermee kan op die systemen de kleur van de karakters worden verlaagd (worden gedimt).

15.2.2.12 NormVideo

Syntax: NormVideo

Ingebouwd voor systemen die een video aansturing gebruiken met attributen. Hiermee kan op die systemen de kleur van de karakters worden teruggezet naar de 'normale' waarde. Dit is de tegenhanger van LowVideo.

15.2.2.13 Randomize

Syntax: Randomize

Initialiseerd de random generator. Als dit niet gebeurd dan zal altijd dezelfde reeks van getallen worden gegenereerd.

15.2.2.14 Move

Syntax: Move(Var1, Var2, Num)

Voert een copie uit in het geheugen van een bepaald aantal bytes. Var1 en Var2 zijn twee
variabelen van een willekeurig type en Num is een Integer expressie. De opdracht
copieert Num bytes, beginnend op het eerste byte van Var1 naar het eerste byte van Var2.
Move zoekt zelf uit of er eventueel een overlapping optreed en zal indien nodig de copieer
opdracht in omgekeerde richting uitvoeren. Hierdoor is het niet nodig om een speciale
MoveLeft en MoveRight te hebben.

15.2.2.15 FillChar

Syntax: FillChar(Var, Num, Value)

Vult een geheugenblok met een bepaalde waarde. Var is een variabele van een
willekeurig type, Num is een Integer expressie en Value is een expressie van het type Byte
of Char. Num bytes, beginnend op het eerste byte van Var, worden gevuld met de waarde
Value.

15.3 Functies

Net als procedures kunnen functies zowel standaard (voor gedefinieerd) als door de
programmeur gedeclareerd zijn.

15.3.1 Functie declaratie

Een functie declaratie bestaat uit een functie hoofd en een blok dat bestaat uit een
declaratie deel en een opdrachtendeel.

Het functie hoofd bestaat uit het gereserveerde woord functie gevolgd door de naam van
de functie, eventueel gevolgd door een formele parameterlijst zoals in hoofdstuk 15.1
beschreven is en wordt afgesloten met een type declaratie van de naam van de functie.

Voorbeelden:

```
function ToetsAanslag: Boolean;

function Bereken(var Waarde: Voorbeeld): Real;

function Kracht(X, Y: Real): Real;
```

Het resultaat van een functie moet een scalair type zijn (Integer, Longint, Boolean, Char,
gedeclareerd scalair of een subrange), een String type of een pointer type.

Het declaratie deel van de functie is hetzelfde als bij een procedure.

Het opdrachtendeel van een functie ziet eruit als een samengestelde opdracht zoals
omschreven in hoofdstuk 6.2. Binnen het opdrachtendeel moet minimaal ÈÈn toekenning
voorkomen aan de functie naam. De laatste toekenning die is uitgevoerd bepaald het
resultaat van de functie. Als de functie aanroep in het opdrachtendeel zelf voorkomt zal
de functie recursief worden aangeroepen. Maar alleen als de {$A-} compiler aanwijzing
vÛÛr het functie hoofd actief was, zie appendix C.

Het volgende voorbeeld toont een functie waarin de som van een rij van integers van I tot
J wordt berekend:

```
function RowSum(I, J: Integer): Integer;

  function SimpleRowSum(S: Integer): Integer;

  begin

    SimpleRowSum:=S*(S+1) div 2;

  end;

begin

  RowSum:=SimpleRowSum(J)-SimpleRowSum(I-1);

end;
```

De functie SimpelRowSum is opgenomen binnen de functie RowSum. SimpelRowSum is
daarom alleen beschikbaar binnen het bereik van RowSum. Het volgende programma is
een klassiek voorbeeld van het gebruikt van een recursieve functie om de faculteit van
een Integer getal te berekenen:

```
{$A+}
```

```
program Factorial;

var Number : Integer;

function Factorial(Value: Integer) : Real;

  begin

    if Value = 0 then

      Factorial:=1

    else

      Factorial:=Value * Factorial(Value-1)

  end;

begin

  Read(Number);

  WriteLn(^M, Number, '! = ', Factorial(Number))

end;
```

Merk op dat een type dat gebruikt wordt in een functie declaratie vooraf moet worden
gedefinieerd als een type naam. Dus de constructie:

```
  function LowerCase(Line: Underline): String[80];
```

is niet toegestaan. In plaats hiervan moet een type worden gedefinieerd en de type naam
moet gebruikt worden in de functie declaratie:

```
  type Str80 = String[80];

  function LowerCase(Line: Underline): Str80;
```

De procedures Read, ReadLn, Write, WriteLn, Str en Val mogen nooit gebruikt worden in
een functie die via een Write of WriteLn worden aangeroepen.

15.3.2 Standaard functies

De volgende standaard (voor gedeclareerde) functies zijn ingebouwd in Turbo Pascal:

1) String bewerkingsfuncties (zie hoofdstuk 8.5)

2) File bewerkingsfuncties (zie hoofdstuk 13.3)

3) Functies voor Pointer bewerkingen (zie hoofdstuk 14.3 en 14.4)

15.3.3 Rekenkundige functies

15.3.3.1 Abs

Syntax : Abs(Num)

Geeft de absolute waarde van Num, het meegegeven argument Num moet van het type Real, Integer of Longint zijn. Het teruggegeven type is hetzelfde als het type van het meegegeven argument.

15.3.3.2 ArcTan

Syntax : ArcTan(Num)

Geeft de hoek terug in radialen waarvan de tangens Num is. Het argument Num moet van het type Real, Integer of Longint zijn. De teruggeven waarde is van het type Real.

15.3.3.3 Cos

Syntax : Cos(Num)

Geeft de cosinus terug van Num. Het argument Num wordt uitgedrukt in radialen en moet van het type Real, Integer of Longint zijn. De teruggegeven waarde is van het type Real.

15.3.3.4 Exp

Syntax : Exp(Num)

Geeft de exponent terug van Num. Het argument Num moet van het type Real, Integer of Longint zijn. De teruggegeven waarde is van het type Real.

15.3.3.5 Frac

Syntax : Frac(Num)

Geeft het deel achter de komma van het getal Num terug. Met andere woorden Frac(Num)= Num-Int(Num). Het argument Num moet van het type Real, Integer of Longint zijn. De teruggegeven waarde is van het type Real.

15.3.3.6 Int

Syntax : Int(Num)

Geeft het gehele deel van Num terug. Als Num groter is dan 0 dan wordt de grootste
Integer kleiner of gelijk aan Num teruggegeven. Als Num kleiner is dan 0 dan wordt de
kleinste Integer groter of gelijk aan Num teruggeven. Het argument Num moet van het
type Real, Integer of Longint zijn. De teruggegeven waarde is van het type Real.

15.3.3.7 Ln

Syntax : Ln(Num)

Geeft de natuurlijke logaritme van Num terug. Het argument Num moet van het type Real,
Integer of Longint zijn. De teruggegeven waarde is van het type Real.

15.3.3.8 Sin

Syntax : Sin(Num)

Geeft de sinus van Num terug. Het argument Num wordt uitgedrukt in radialen en moet
van het type Real, Integer of Longint zijn. De teruggegeven waarde is van het type Real.

15.3.3.9 Sqr

Syntax : Sqr(Num)

Geeft het kwadraat terug van Num, met andere woorden Num * Num. Het argument Num
moet van het type Real, Integer of Longint zijn. De teruggegeven waarde is van hetzelfde
type als het argument.

15.3.3.10 Sqrt

Syntax : Sqrt(Num)

Geeft de wortel terug van Num. Het argument Num moet van het type Real, Integer of
Longint zijn. De teruggegeven waarde is van het type Real.

15.3.4 Scalaire functies

15.3.4.1 Pred

Syntax : Pred(Num)

Geeft de voorganger van Num (indien deze bestaat) terug. Num is van een willekeurig
scalair type.

15.3.4.2 Succ

Syntax : Succ(Num)

Geeft de opvolger van Num (indien deze bestaat) terug. Num is van een willekeurig
scalair type.

15.3.4.3 Odd

Syntax : Odd(Num)

Geeft de booleaanse waarde True terug als Num een oneven getal is en False als Num een
even getal is. Num moet van het type Integer of Longint zijn.

15.3.5 Conversie functies

De conversie functies zetten een scalair type om naar een ander scalair type. Naast de
hierna genoemde functies kan ook type conversie zoals in hoofdstuk 7.3 beschreven is
worden gebruikt.

15.3.5.1 Chr

Syntax: Chr(Num)

Geeft het karakter terug met de ordinale waarde Num. Chr(65) geeft het karakter 'A'.

15.3.5.2 Ord

Syntax: Ord(Varx)

Geeft het ordinale nummer terug van de waarde Varx. In de set gedefinieerd door het type
Varx. Ord(Varx) is gelijk aan Integer(Varx). Varx mag van elk scalair type zijn
behalve Real. Het resultaat van Ord is van het type Integer.

15.3.5.3 Round

Syntax: Round(Num)

Geeft de waarde terug van Num afgerond naar de naast gelegen integer. Als Num groter
dan of gelijk aan 0 is, dan geldt Round(Num)=Trunc(Num+0.5) en als Num kleiner
dan 0 is, dan geldt Round(Num)=Trunc(Num-0.5). Num moet van het type Real zijn en het resultaat is van het type Integer.

15.3.5.4 Trunc

Syntax: Trunc(Num)

Als Num groter is dan 0, dan zal Trunc de grootste Integer kleiner of gelijk is aan Num teruggeven. Als Num echter kleiner is dan 0, dan zal Trunc de kleinste Integer die kleiner of gelijk is aan Num teruggeven. Num moet altijd van het type Real zijn en het resultaat is van het type Integer of Longint.

15.3.6 Uitgebreide standaard functies

15.3.6.1 Hi

Syntax: Hi(Num)

Deze functie heeft twee mogelijke resultaten. Afhankelijk van het type van Num is dit een Integer een Longint. Als Num een Integer is geeft de functie het meest significante byte van Num terug. Als Num een Longint is geeft de functie het meeste significantie Integer deel van Num terug of wel de hoogste 16 bits.

Het resultaat van de functie Hi is altijd van het type Integer.

15.3.6.2 KeyPressed

Syntax: KeyPressed

Geeft de booleaanse waarde True terug als er een toets op het toetsenbord is ingedrukt en als er geen toets is ingedrukt de waarde False. Het resultaat wordt verkregen door een aanroep van de console status routine uit het besturingssysteem.

15.3.6.3 Lo

Syntax: Lo(Num)

Deze functie heeft twee mogelijke resultaten. Afhankelijk van het type van Num is dit een Integer of een Longint. Als Num een Integer is geeft de functie het minst significante byte van Num terug. Als Num een Longint is geeft de functie het minst significantie Integer deel van Num terug of wel de laagste 16 bits.

Het resultaat van functie Lo is altijd van het type Integer.

15.3.6.4 Random

Syntax: Random

Geeft een willekeurig getal terug dat groter of gelijk is aan 0 en kleiner dan 1. Het resultaat is van het type Real.

15.3.6.5 Random(num)

Syntax: Random(Num)

Geeft een willekeurig nummer dat groter of gelijk is aan 0 en kleiner dan Num. Zowel het
resultaat als Num zijn van het type Integer.

15.3.6.6 Paramcount

Syntax: Paramcount

Deze Integer functie geeft als resultaat het aantal parameters dat is meegegeven aan het
programma vanaf de command line. Spaties en tabs dienen als scheidingsteken.

15.3.6.7 ParamStr

Syntax: ParamStr(N)

Deze String functie geeft de N-de parameter van de command line buffer terug.

15.3.6.8 SizeOf

Syntax: SizeOf(VarX)

Deze functie geeft het aantal bytes terug dat de variabele VarX of het type VarX in het
geheugen in beslag neemt. Het resultaat is van het type Integer. Als VarX van het type
Longint is, is het resultaat 4.

15.3.6.9 Swap

Syntax: Swap(Num)

Deze functie wisselt het minst significante deel en het meest significante deel van Num
om. Als Num van het type Integer is, dan worden het hoge en het lage byte verwisselt. Is
Num van het type Longint, dan worden de hoge 16 bits verwisselt met de lage 16 bits.

Het resultaat blijft in beide gevallen van hetzelfde type.

Voorbeeld:

```
  Swap($1234)      levert $3412

  Swap($12345678)  levert $56781234
```

15.3.6.10 UpCase

Syntax: UpCase(Ch)

Deze functie verandert een kleine letter in een hoofdletter, als Ch geen kleine letter is dan wordt Ch ongewijzigd als resultaat teruggegeven. Ch is van het type Char en het resultaat ook.

15.4 Voorwaartse referenties

Het gereserveerde woord forward wordt gebruikt indien men een functie of procedure wil aanroepen die op het moment van de aanroep nog niet bekend is bij de compiler. Het functie- of procedure hoofd wordt gevolgd door het gereserveerde woord forward.

Het opdrachtendeel dat bij deze forward declaratie hoort hoeft dan pas later te worden gecompileerd. De forward declaratie is hetzelfde als een normaal programma- of functie hoofd.

Het opdrachtendeel dat pas veel later voorkomt moet wel worden voorafgegaan door een functie- of procedure hoofd, hierbij mogen geen parameters aanwezig zijn.

Voorbeeld:

```
program Catch22;

var X: Integer;



function Up(var I: Integer): Integer; forward;



function Down(var I: Integer): Integer;

begin

  I:=I div 2;

  WriteLn(I);

  if I<>1 then I:=Up(I);

end;



function Up;

begin
```

```
  while I mod 2<>0 do

  begin

    I:=I*3+1;

    WriteLn(I);

  end;

  I:=Down(I);

end;


begin

  Write('Enter any integer: ');

  ReadLn(X);

  X:=Up(X);

  Write('Ok. Program stopped again.');

end.
```

Als dit programma uitgevoerd wordt en men bijvoorbeeld 6 invoert, dan krijgt men als resultaat:

```
3

10

5

16

8

4

2

1

Ok. Program stopped again.
```

Het bovenstaande programma is een gecompliceerdere versie van het volgende programma:

```
program Catch222;

var X: Integer;

begin

  Write('Enter any integer:');

  ReadLn(X);

  while X<>1 do

  begin

    if X mod 2=0 then X:=X div 2 else X:=X*3+1;

    WriteLn(X);

  end;

  Write('Ok. Program stopped again.');

end.
```

Misschien interesseert het u, dat van dit zeer kleine en eenvoudige programma niet bewezen kan worden dat het voor elke ingegeven heel getal stopt!

16. Overlay systeem

Het overlay systeem biedt de mogelijkheid om programma's te maken die veel groter zijn dan het werkgeheugen van de computer. De gebruikte techniek bestaat uit het verzamelen van deelprogramma's (procedures en/of functies) in ÈÈn of meer files, apart van het hoofdprogramma. Deze delen worden dan automatisch tijdens de uitvoer van het programma geladen. Voor de verschillende delen kan dan hetzelfde geheugengebied worden gebruikt.

Figuur 16 - 1 toont een programma dat gebruik maakt van een overlay file met daarin vijf overlay deelprogramma's. De vijf deelprogramma's zijn verzameld in ÈÈn overlay groep.
Alle deelprogramma's gebruiken hetzelfde stuk van het werkgeheugen.

Figuur 16 - 1 Principe van het overlay systeem

Zodra een overlay routine wordt aangeroepen, wordt deze automatisch geladen in het geheugengebied dat voor die overlay is gereserveerd. Het gereserveerde geheugen is even groot als de grootste routine uit de overlay file. De winst die u behaald uit het gebruik van een overlay bestaat dus uit de som van alle routines min de grootste routine uit de overlay file.

In het voorgaande voorbeeld is de tweede routine uit de overlay file het grootste. De tweede routine bepaald daarom het geheugenformaat van het gereserveerde gebied. Als de tweede routine is geladen, is dus al het gereserveerde geheugen in gebruik.

Figuur 16 - 2 Het grootste overlay subprogramma is geladen.

De kleinere deelprogramma's worden in hetzelfde deel van het geheugen geladen. Elke routine begint op het eerste adres van het gereserveerde overlay gebied. De kleinere routines gebruiken natuurlijk maar een deel van het gereserveerde gebied, het resterende deel blijft ongebruikt.

Figuur 16 - 3 Een kleiner overlay deelprogramma is geladen.

Omdat procedure 1, 3, 4 en 5 in hetzelfde gebied worden uitgevoerd als routine 2, is er in het hoofdprogramma geen extra ruimte voor deze routines nodig. In Turbo Pascal versie 3.0a was het niet mogelijk dat deze routines elkaar konden aanroepen. In versie 3.3 kunnen overlay routines elkaar aanroepen, net alsof het normale procedures en functies zijn, zowel binnen dezelfde overlay als aanroepen naar andere overlays. Recursie is dus ook mogelijk, mits het {$A-} compiler directive wordt gebruikt.

Er kunnen veel meer routines in een overlay file aanwezig zijn dan de vijf routines in dit voorbeeld. Het is zelfs mogelijk dat de overlay file veel groter is dan het hoofdprogramma. De ruimte die binnen het hoofdprogramma wordt gereserveerd zal toch nooit groter zijn dan de grootste routine uit de overlay file.

Het voordeel van het gebruik van een overlay is een kleinere programma code. Er is echter ook een nadeel. Tijdens de uitvoering van uw programma is er tijd nodig om de juiste overlay routine van disk in te lezen in het werkgeheugen. Met de juiste planning zoals deze verderop in dit hoofdstuk staat beschreven, is deze extra tijd echter te verwaarlozen.

16.1 Overlay maken

Overlay deelprogramma's worden automatisch gemaakt door het toevoegen van het gereserveerde woord overlay aan de declaratie van een procedure of functie.

```
   overlay procedure Initializatie;
```

en

```
   overlay function TijdVanDeDag: Tijd;
```

Als de compiler deze declaratie tegen komt wordt de code niet langer naar het hoofdprogramma uitgevoerd, maar naar de aparte overlay file. De naam van deze overlay file zal hetzelfde zijn als de naam van het hoofdprogramma en de extensie van de naam zal bestaan uit een drie cijferig nummer tussen 000 en 099.

Opeenvolgende overlay routines zullen bij elkaar gevoegd worden in ÈÈn overlay file. Met andere woorden, zolang er geen procedures of functies tussen staan zonder de toevoeging overlay zullen alle overlay routines in ÈÈn overlay file terecht komen.

Voorbeeld 1:

```
overlay procedure Een;

begin

   :

end;


overlay procedure Twee;

begin

   :

end;


overlay procedure Drie;

begin

   :

end;
```

Deze drie overlay procedures zullen bij elkaar in ÈÈn overlay file worden geplaatst. Als dit de eerste groep van overlay routines is dan zal de extensie van de overlay file .000 zijn.

De drie overlay procedures in het volgende voorbeeld zullen worden geplaatst in de overlay files met de extensies .000 en .001. Dit gebeurd omdat de declaratie van de niet overlay routine Teller de beide overlay blokken van elkaar scheidt.

De scheiding van overlay blokken kan met iedere declaratie plaatsvinden. Het zou bijvoorbeeld ook mogelijk zijn de scheiding te bewerkstelligen door de declaratie van een type.

Voorbeeld 2:

```
overlay procedure Een;
```

```
begin
  :
end;


overlay procedure Twee;
begin
  :
end;


procedure Teller;
begin
  :
end;


overlay procedure Drie;
begin
  :
end;
```

Voor elke groep overlay routines wordt een apart overlay gebied binnen het geheugen
gereserveerd en de volgende overlay files zullen ontstaan:

Figuur 16 - 4 Meerdere overlay files

16.2 Overlay files binnen een overlay file (geneste overlays)

Overlay deelprogramma's kunnen binnen elkaar vallen. Met andere woorden, een

overlay programma kan binnen zichzelf een tweede overlay deel bevatten.

Voorbeeld 3:

```
program OverlayDemo;
  :
```

```
  :

overlay procedure Een;

begin

  :

end;


overlay procedure Twee;

  overlay procedure Drie;

  begin

    :

  end;

begin

  :

end;

  :
```

In dit voorbeeld worden twee overlay files gemaakt. File .000 bevat de routines Een en Twee. Binnen het hoofdprogramma wordt de ruimte gereserveerd voor de grootste van deze twee. Overlay file .001 bevat de routine Drie en de ruimte die voor deze routine is gereserveerd valt binnen het gebied dat was gereserveerd voor routine Twee.

Figuur 16 - 5 Geneste overlay files.

16.3 Automatisch overlay beheer

Een routine uit een overlay file wordt alleen in het geheugen geladen als er aanroep vanuit het overige programma plaatsvind. Bij iedere aanroep wordt automatisch gekeken of de gewenste routine in het geheugen aanwezig is en zo nodig wordt de routine ingeladen vanuit de juiste overlay file.

In deze versie van Turbo Pascal kan men zelf een procedure definiÎren die de gewenste
overlay routine inlaad. Dit geeft de mogelijkheid om b.v. een overlay file vooraf in de
memory mapper in te laden om daarna vanuit de mapper de overlay routines in te laden.

16.4 Plaats van overlay files

Tijdens het compileren van een programma worden de files op dezelfde drive geschreven
als het hoofd programma, de .COM file.

Bij het uitvoeren van een programma verwacht het programma de overlay files te vinden
op de default drive.

16.5 Efficient gebruik van overlays

Het gebruik van overlay files in een programma heeft enkele nadelen. Ten eerste is er de
extra tijd die nodig is tijdens de uitvoer van het programma. Het programma moet de
routines van disk inladen, dit kost natuurlijk wat tijd. Daarnaast is er binnen het
programma wat extra code nodig om er voor te zorgen dat de juiste routine op de juiste
plaats wordt ingelezen.

Om een programma niet te veel te vertragen is het verstandig om er voor te zorgen dat een
overlay routine niet te veel wordt aangeroepen of er voor te zorgen dat routines die vaak
worden aangeroepen niet uit dezelfde overlay file afkomstig zijn. Daarnaast is er
natuurlijk veel verschil of de routine die moet worden ingelezen afkomstig is van een
diskdrive, een harddisk of een ramdisk.

Om zo veel mogelijk ruimte in het hoofdprogramma te besparen is het aan te bevelen zo
veel mogelijk routines in ÈÈn overlay file te plaatsen. Puur vanuit besparings oogpunt
gezien is het plaatsen van veel routines in een overlay file het meest economische. Dit
moet echter wel worden afgewogen tegen de benodigde extra tijd zoals hierboven is
besproken.

16.6 Beperkingen door overlay files

16.6.1 Data gebied

Overlay deelprogramma's die in dezelfde groep vallen gebruiken allemaal hetzelfde geheugen en kunnen dus niet gelijktijdig aanwezig zijn. Om deze reden kunnen ze ook niet elkaar aanroepen. Daarnaast gebruiken ze ook allemaal hetzelfde data gebied (wat op zich ook weer geheugenbesparend werkt).

In programma voorbeeld 1 (zie pagina 99) mag geen van de overlay routines een andere aanroepen. In programma voorbeeld 2 (zie pagina 100) mag procedure Drie wel procedure Een en Twee aanroepen omdat deze in een ander overlay gebied binnen het hoofdprogramma vallen.

16.6.2 Forward declaratie

Een overlay routine mag niet met een forward declaratie worden gedeclareerd. Deze beperking is echter eenvoudig te omzeilen door een kleine procedure te declareren die de overlay procedure aanroept.

16.6.3 Recursie

Overlay routines mogen recursief zijn in Turbo Pascal versie 3.3. Men MOET er voor zorgen dat er geen absolute code wordt gegenereerd, d.w.z. het compiler directive {$A-} moet voor de betreffende recursieve overlay routines worden geplaatst. Gebeurt dit niet, dan wordt de code van de vorige overlay routine niet herstelt.

16.6.4 Looptijdfouten

Fouten die zich tijdens de uitvoering voordoen worden op dezelfde manier ontdekt als alle overige fouten. U krijgt het adres terug van de positie van de fout. Dit adres is echter een adres binnen het overlay gebied. Omdat er meerdere routines binnen dit overlay gebied vallen is het onmogelijk om te vermelden in welke routine binnen de overlay file de fout optreedt.

Als u het gevonden adres terugzoekt met de foutzoek optie (/F) uit de compiler zal de compiler bij het eerste keer dat dat adres door een routine gebruikt wordt de melding geven dat de fout is gevonden. Dit adres kan echter al in gebruikt zijn door een eerdere routine uit die overlay file. De melding wordt dan dus bij de verkeerde routine gegeven.

Deze fout is niet te omzeilen. Wel is vaak te beoordelen in welke routine de fout optreedt.
U kunt deze routine als eerste routine in de overlay file zetten en de fout zal dan wel goed
worden gevonden.

Het meest verstandige is alleen routines in een overlay te plaatsen die volledig zijn
uitgetest en geen fouten meer bevatten.

16.7 Laden van overlays

In Turbo Pascal versie 3.3 is het mogelijk om als gebruiker een eigen laad procedure te
maken voor de overlays. Deze procedure wordt door Turbo Pascal aangeroepen als de
nieuw te laden overlay een andere is dan de huidige.

Deze laad procedure heeft de volgende declaratie:

```
  procedure OverlayHandler(Address: Integer;
                           Pos: Longint;
                           DataLength: Integer;
                           OverlayNum: Integer;
                           var Name: Str63);
```

Address is het adres waar de code van de overlay moet komen te staan.

Pos is de positie binnen de overlay file waar de code staat, deze is byte-georiÎnteerd.

DataLength is de lengte van de te lezen overlay code.

OverlayNum is het nummer van de overlay file en varieert van 00 t/m 99.

Name is een String waarin als extra informatie nogeens de door Turbo Pascal toegekende
naam van de overlay staat.

Een goed voorbeeld voor het gebruik van de laad procedure is b.v. het vooraf inlezen van
ÈÈn of meerdere overlay files in de memory mapper welke m.b.v. de GIOS procedures
SetChannel en ReadMem benaderd kunnen worden.

Voorbeeld:

```
procedure OverlayHandler(Address: Integer;

                         Pos: Longint;

                         DataLength: Integer;

                         OverlayNum: Integer;

                         var Name: Str63);

begin

   SetChannel(0, OverlayBaseAddress[OverlayNum] + Pos);

   ReadMem(0, Address, DataLength);

end;
```

In dit voorbeeld wordt er vanuit gegaan dat de startposities van de overlays in het array
OverlayBaseAddress staan.

17. Systeem

Dit hoofdstuk beschrijft de speciale mogelijkheden van de compiler om er efficiÎnt
gebruik van te kunnen maken. Aan het eind worden er een aantal technische mogelijkheden uitgelegd voor de ervaren programmeur, zoals machinecode routines.

17.1 Compiler opties

Hieronder volgt een beschrijving van de mogelijke compiler opties, deze beschrijving wordt weergegeven als er geen opties worden gegeven of als er een ongeldige optie wordt meegegeven. Deze opties waren voorheen instelbaar via het menu van Turbo Pascal 3.0.

```
Z80 TURBO Pascal compiler,            Version 3.3

Copyright (C) MSX computer club Enschede, '93-'96

Syntax: TURBO <filename> [/C|/H] [/Sxxxx] [/Exxxx] [/Fxxxx]

              [/R[-|<file>]]

/C            compiles <filename> to COM-file

/H            compiles <filename> to CHAIN-file

/Sxxxx        code start address (hexadecimal), min. 2DB5

/Exxxx        data end address (hexadecimal), max. D3E8

/Fxxxx        find runtime error at address xxxx (hexadecimal)

/R<file>      generates this error file with a compiler error

/R-           generates no error file
```

17.1.1 Compileren naar COM-file

Syntax: /C

Met deze optie wordt er door de compiler een COM-file gecreÎerd. Dit is de default instelling.

17.1.2 Compileren naar CHAIN-file

Syntax: /H

Met deze optie wordt er door de compiler een CHN-file gecreÎerd. Deze kan worden gestart d.m.v. de procedure Chain.

17.1.3 Startadres

Syntax: /Sxxxx

Het startadres specificeert het adres van de eerste byte van de code (hexadecimaal). Normaal gesproken is dit het eindadres van de Pascal bibliotheek plus 1, maar het kan veranderd worden, om zo bijvoorbeeld ruimte maken voor absolute variabelen, die gedeeld worden door verschillende met behulp van het Chain commando

aaneengeschakelde programma's.

17.1.4 Eindadres

Syntax: /Exxxx

Het eindadres specificeert het hoogste adres dat tot beschikking van het programma staat
(hexadecimaal). De waarde na max. (zie boven) is de standaard waarde.

Als gecompileerde programma's in een andere omgeving gerund worden, kan het
eindadres aangepast worden aan de TPA grootte van dat systeem. Als u vooruit weet dat
uw programma op veel verschillende computers moet runnen, is het wijs om deze waarde
relatief laag te zetten, bijvoorbeeld /EC100 (48K).

17.1.5 Vinden van looptijdfouten

Syntax: /Fxxxx

Als er een looptijdfout optreedt wordt door Pascal de code van de fout weergegeven en de
waarde van de programma teller (PC) op het moment van de fout.

Bijvoorbeeld:

```
Run-time error NN, PC=addr

Program aborted
```

Om de plaats van de fout in de brontekst te vinden, moet u de bron code opnieuw vertalen
met dezelfde start- en eindadressen en extra de optie /F meegeven met als argument het
adres (addr) dat bij de looptijdfout is weergegeven. Als de fout wordt gevonden, dan
wordt er een melding gemaakt op het scherm en tevens wordt er een file gemaakt met de
extensie .ERR waarin nog eens alle informatie over de gevonden fout staat (zie de /R
optie).

17.1.6 De error file

Syntax: /R-

of

Syntax /R<filenaam>

Bij iedere compiler fout wordt er standaard altijd een error file aangemaakt. Deze error
file bestaat uit 5 regels die alle informatie bevatten over de positie van de fout. De naam
van deze error file wordt standaard samengesteld uit de filenaam van het te compileren
pascal programma met als standaard extensie .ERR. Deze twee opties bieden de
mogelijkheid om de error file een andere naam te geven of om de error file helemaal niet
aan te maken.

De optie /R- zorgt ervoor dat er helemaal geen error file wordt gemaakt.

Bij optie /R<filenaam> kan men een eigen filenaam opgeven. B.v. /Rfout.txt
maakt bij een fout de file 'fout.txt' aan.

De aangemaakte error file bestaat altijd uit 5 regels en deze zijn als volgt ingedeeld:

1) De naam van de pascal file inclusief een eventuele padnaam

2) Het regelnummer in de genoemde pascal file

3) Het kolumnummer

4) Het nummer van de fout

5) De omschrijving van het foutnummer.

Voorbeeld:

PASCAL\TEST.PAS

5

4

41

Unknown identifier or syntax error

17.2 Standaard Identifiers

De volgende standaard identifiers zijn uniek voor de CP/M implementatie van Turbo
Pascal:

```
  Bios    Bdos    RecurPtr

  BiosHL  BdosHL  StackPtr
```

17.2.1 Chain en Execute

Turbo Pascal levert twee standaard procedures: Chain en Execute die het mogelijk
maken om programma's te activeren vanuit een Pascal programma. De syntax voor de
procedure aanroepen is:

```
  Chain(FilVar)

  Execute(FilVar)
```

FilVar is een file variabele van een willekeurig type, die daarvoor een disk file
toegewezen heeft gekregen met behulp van de procedure Assign. Als de file bestaat,
wordt het in het geheugen geladen en uitgevoerd.

De Chain procedure wordt alleen gebruikt om speciale .CHN files uit te voeren. Dit zijn
files die gecompileerd zijn met de /H optie, zie 17.1.2 "Compileren naar CHAIN-file".
Een CHAIN-file bevat alleen maar code, geen Pascal bibliotheek. Het wordt in het
geheugen geladen en gestart op het startadres van het huidige programma (dat is het adres
gespecificeerd toen het huidige programma werd gecompileerd, zie 17.1.3 "Startadres").
Het gebruikt dan de Pascal bibliotheek die al in het geheugen aanwezig is. Dus: het
huidige programma en het ge-Chain-de programma moeten hetzelfde startadres hebben.

De Execute procedure kan gebruikt worden om iedere .COM file uit te voeren, dus
iedere file die uitvoerbare code bevat. Dit kan een file zijn die door Turbo Pascal
gecreÎerd is met de /C optie, zie 17.1.1 "Compileren naar COM-file". Het programma

wordt geladen en uitgevoerd vanaf adres $100, zoals de CP/M standaard het voorschrijft.

Als de file niet bestaat op de disk, treed er een I/O fout op. Deze fout wordt afgehandeld
zoals beschreven in paragraaf 13.14 "I/O controle". Als de I compiler directive passief is
({$I-}), wordt verder gegaan met het uitvoeren van het programma, bij het statement
volgend op het Chain of Execute statement dat gefaald heeft en de waarde van
IOResult moet opgevraagd worden voor er weer I/O wordt gedaan.

Data kan overgedragen worden van het huidige programma aan het ge-Chain-de of ge-
Execute programma door gedeelde globale variabelen of door absolute variabelen.

Om overlapping te verzekeren, moeten gedeelde globale variabelen gedeclareerd worden
als de eerste variabelen in beide programma's en in dezelfde volgorde. Verder moeten
beide programma's met dezelfde start- en eindadressen. Als aan deze condities voldaan is
zullen de variabelen in beide programma's op hetzelfde adres geplaatst worden en omdat
Turbo Pascal variabelen geen begin waarde geeft, kunnen ze gedeeld worden.

Voorbeeld:

Het programma MAIN.COM:

```
program Main;

var

  Txt    : String[80];

  CntPrg : File;

begin

  Write('Geef een tekst: ');

  ReadLn(Txt);

  Assign(CntPrg,'ChrCount.chn');

  Chain(CntPrg);
```

```
end.
```

Het programma CHRCOUNT.CHN:

```
program ChrCount;
var
  Txt :   String[80];
  NoOfChar,
  NoOfUpc,
  I   :   Integer;
begin
  NoOfUpc := 0;
  NoOfChar:=Length(Txt);
  for I:=1 to Length(Txt) do
    if Txt[I] in ['A'..'Z'] then NoOfUpc:=Succ(NoOfUpc);
  Write('Aantal character in de invoer: ',NoOfChar);
  WriteLn('. Aantal hoofdletter: ',NoOfUpc,'.');
end.
```

17.3 Bestanden

17.3.1 Namen van bestanden:

Een naam van een bestand bestaat onder CP/M uit 1 tot 8 letters of cijfers, optioneel gevolgd door een punt en een bestand type bestaande uit drie letters of cijfers:

Drive: Naam.Type

Voorbeeld:

A:\TEST.1AF

17.3.2 Text bestanden

De Seek en Flush procedures en de FilePos en FileSize functies zijn niet toepasbaar op tekst bestanden.

17.3.3 Absolute variabelen

Variabelen kunnen gedeclareerd worden op een bepaald adres in het geheugen en worden
dan absoluut genoemd. Dit wordt gedaan door het gereserveerde woord absolute en een
adres uitgedrukt door een Integer constante toe te voegen aan de declaratie.

Voorbeeld:

```
var

  IOByte : Byte        absolute $0003;

  CmdLine: String[127] absolute $0080;
```

Absolute kan ook gebruikt worden om een variabele "bovenop" een andere variabele te
declareren. De variabelen starten dan op hetzelfde geheugen adres. Als absolute gevolgd
wordt door een variabele (of parameter) naam, zal de nieuwe variabele op het beginadres
van die andere variabele (of parameter) beginnen.

Voorbeeld:

```
var

  Str    : String[32];

  StrLen : Byte absolute Str;
```

De bovenstaande declaratie zegt dat de variabele StrLen moet beginnen op hetzelfde
adres als de variabele Str en omdat het eerste byte van een string de lengte van die string
bevat, bevat StrLen de lengte van Str. Merk op dat er slechts 1 variabele
gespecificeerd mag worden in een absolute declaratie, dat betekent dat de volgende
declaratie niet is toegestaan:

```
  Ident1,Ident2 : Integer absolute $8000;
```

Voor verdere details over de geheugen allocatie voor variabelen wordt verwezen naar 17.14 "Interne Data Formaten" en 17.14.5 "De heap en de stack".

17.4 Addr functie

Syntax: Addr(Name)

Geeft het adres in het geheugen van de eerste byte van een type, variabele, procedure of een functie met de gegeven Name. Als Name een array is mag er ook een index gespecificeerd worden en als Name een record is mag er een veld geselecteerd worden.
De waarde die teruggegeven wordt is van het type Integer.

17.5 Voorgedefinieerde Arrays

Turbo Pascal biedt twee voorgedefinieerde arrays van het type Byte. Deze heten Mem en Port en ze worden gebruikt voor directe toegang tot het CPU geheugen en de I/O poorten.

17.5.1 Mem Array

De voorgedefinieerde array Mem wordt gebruikt voor toegang tot het geheugen. Elk component van dit array is een Byte en de index correspondeert met een geheugenadres.
De index is van het type Integer. Als een waarde wordt toegewezen aan een component van het Mem array, wordt het opgeslagen op het adres dat is opgegeven in de index expressie. Als de Mem array wordt gebruikt in een expressie, wordt het Byte op het adres dat door de index wordt gespecificeerd gelezen.

Voorbeelden:

```
  Mem[WsCursor]    := 2;

  Mem[WsCursor+1]  := $1B;

  Mem[WsCursor+2]  := Ord(' ');

  IOByte           := Mem[3];

  Mem[Addr+Offset] := Mem[Addr];
```

17.5.2 Port Array

Het Port array wordt gebruikt voor toegang tot de I/O poorten van de Z80. Elk element

van dit array representeert een I/O poort. Omdat I/O poorten geadresseerd worden door 8-bits adressen, is de index van het type Byte. Als een waarde wordt toegekend aan een element van het Port array, wordt de waarde naar de aangegeven poort geschreven. Als een element uit het array wordt gebruikt in een expressie, wordt de waarde gelezen uit de gespecificeerde poort.

Het gebruikt van het Port array is beperkt tot toekenning van waarden en gebruik in expressies. Dat wil zeggen dat een Port element niet gebruikt mag worden als variabele parameter voor functies en procedures. Verder zijn operaties die verwijzen naar het gehele Port array (verwijzingen zonder gebruik van index) niet toegestaan.

17.6 Array index optimalisatie

Het X compiler directive staat de programmeur toe om aan te geven of array indices geoptimaliseerd moeten worden met betrekking tot de uitvoer snelheid of op lengte van de code. De standaard instelling voor deze directive is {$X+} welke uitvoer snelheid geoptimaliseerd betekend. Als de optie passief is ({$X-}) wordt de grootte van de code geminimaliseerd.

17.7 With opdrachten

De standaard maximum diepte van het nesten van with statements is 2, maar de W directive kan gebruikt worden om deze waarde te veranderen tussen de 0 en de 9. Voor ieder blok gebruiken with opdrachten 2 bytes voor elk toegestaan niveau. Beperken van het aantal geneste with opdrachten kan de grootte van het data gebied in programma's met veel subprogramma dus sterk beÔnvloeden.

Het is sterk aan te raden om, als men het with statement niet gebruikt, directive {$W0} op te nemen. Dit scheelt per gedefinieerde procedure of functie 4 bytes voor de standaard instelling van 2.

17.8 Pointer gerelateerde elementen

17.8.1 MemAvail

De standaard functie MemAvail is er om de beschikbare ruimte op de heap te bepalen op ieder gewenst moment. Het resultaat is een Integer waarde en als er meer dan 32767 bytes beschikbaar zijn, retourneert MemAvail een negatieve waarde. Het correcte aantal bytes beschikbaar op de heap is dan gelijk aan 65535.0 + MemAvail. Merk op dat een Real constante wordt gebruikt om een Real resultaat te verkrijgen, omdat het resultaat groter is dan MaxInt. Geheugen beheer wordt uitvoeriger behandeld in 17.14.5 "De heap en de stack".

17.8.2 Pointers en Integers

De standaard functies Ord en Ptr geven directe controle over het adres dat een pointer bevat. Ord geeft het adres van de pointer als een Integer en Ptr converteerd een Integer naar een pointer die compatibel is met alle pointer typen.

Deze functies zijn bijzonder waardevol in de handen van een ervaren programmeur, omdat ze een pointer naar alles in het geheugen kunnen laten wijzen. Als er onbedachtzaam gebruikt van wordt gemaakt kunnen ze echter erg gevaarlijk zijn, omdat er dynamische variabelen gemaakt kunnen worden die andere variabelen of zelfs code overschrijven.

17.9 CP/M Functie aanroepen

Om de CP/M BDOS en BIOS routines te kunnen aanroepen, heeft Turbo Pascal twee standaard procedures: BDos, Bios en vier standaard functies: BDos, BDosHL, Bios en BiosHL.

17.9.1 BDos procedure en functie

```
Syntax:    BDos(Func {, Param } )
```

De BDos procedure wordt gebruikt om een CP/M BDOS routine aan te roepen. Func en Param zijn Integer expressies. Func geeft het nummer van de aangeroepen routine aan en wordt in het Z80 register C gezet. Param is optioneel en geeft een parameter aan die in het Z80 registerpaar DE gezet wordt voordat adres $0005 wordt aangeroepen om de BDos routine te starten.

De BDos functie wordt aangeroepen net als de procedure, maar geeft een Integer waarde
terug. Dat is de waarde die de BDos routine plaatst in het register A.

17.9.2 BDosHL Functie

Syntax: BDosHL(Func, {, Param })

Deze functie is exact gelijk aan de BDos functie die hierboven beschreven is, behalve dat
het resultaat de waarde is van het Z80 register paar HL.

17.9.3 Bios procedure en functie

Syntax: Bios(Func {, Param })

De Bios procedure wordt gebruikt om BIOS routines aan te roepen. Func en Param
zijn Integer expressies. Func geeft het routine nummer van de aan te roepen BIOS
routine aan (0 betekent WBOOT routine, 1 de CONST routine etc.) Het adres van de
routine is Func * 3 plus het adres van WBOOT dat staat op de adressen $0001 en
$0002. Param is optioneel en geeft een parameter weer die in het Z80 registerpaar BC
gezet wordt voordat de aanroep wordt uitgevoerd.

De Bios functie wordt net als de procedure aangeroepen, maar retourneerd een Integer
die overeenkomt met het Z80 register A na de terugkeer van de BIOS routine.

17.9.4 BiosHL functie

Syntax: BiosHL(Func {, Param })

Deze functie is precies hetzelfde als de Bios functie die hierboven beschreven is,
behalve dan, dat het resultaat de waarde van het Z80 registerpaar HL is.

17.10 Definieerbare I/O drivers

Voor sommige applicaties is het praktisch voor de programmeur om zijn eigen I/O
drivers (routines die de invoer en uitvoer van karakters van en naar externe apparaten
verzorgen) te kunnen definiÎren. De volgende drivers zijn deel van de Turbo Pascal
omgeving en worden gebruikt door de standaard I/O (hoewel ze niet beschikbaar zijn als
standaard procedures en functies):

```
  function  ConSt : Boolean;
```

```
function  ConIn : Char;

procedure ConOut(Ch : Char);

procedure LstOut(Ch : Char);

procedure AuxOut(Ch : Char);

function  AuxIn : Char;

procedure UsrOut(Ch : Char);

function  UsrIn : Char;
```

De ConSt routine wordt aangeroepen door de functie KeyPressed, de ConIn en
ConOut routines worden gebruikt door de CON:, TRM: en KBD: apparaten, de LstOut routine wordt gebruikt door het LST: apparaat, de AuxOut en AuxIn routines
worden gebruikt door het AUX: apparaat en de UsrOut en UsrIn routines door het
USR: apparaat.

Als beginwaarde gebruiken deze drivers de corresponderende BIOS routines van het

CP/M systeem. Dat wil zeggen: ConSt gebruikt CONST, ConIn gebruikt CONIN, ConOut gebruikt CONOUT, LstOut gebruikt LIST, AuxOut gebruikt PUNCH, AuxIn gebruikt READER, UsrOut gebruikt CONOUT en UsrIn gebruikt CONIN. Dit kan echter door de programmeur gewijzigd worden door het adres van een
zelfgemaakte driver routine toe te wijzen aan een van de volgende standaard variabelen:

Variabele

Bevat het adres van de

ConStPtr

ConSt functie

ConInPtr

ConIn functie

ConOutPtr

ConOut procedure

LstOutPtr

LstOut procedure

AuxOutPtr

AuxOut procedure

AuxInPtr

AuxIn functie

UsrOutPtr

UsrOut procedure

UsrInPtr

UsrIn functie

Tabel 17 - 1
Een door de gebruiker gedefinieerde driver procedure of driver functie moet aan de bovenstaande definities voldoen. Een ConSt driver moet een Boolean functie zijn en een ConIn moet een Char functie zijn, etc.

17.11 Externe sub-programma's

Het gereserveerde woord external wordt gebruikt om externe procedures en functies te declareren. Typisch zijn dit procedures en functies die in machinetaal geschreven zijn.

Een extern subprogramma heeft geen declaratie gedeelte en geen opdrachten deel. Alleen de kop wordt gespecificeerd, direct gevolgd door het gereserveerde woord external en een Integer konstante die het adres van het subprogramma definieert:

```
procedure DiskReset; external $EC00;

function  IOStatus: Boolean; external $D123;
```

Parameters kunnen doorgegeven worden aan deze externe routines en de syntax van de aanroepen naar zulke routines is precies hetzelfde als de aanroepen naar gewone procedures en functies:

```
procedure Plot(X,Y ; Integer); external $F003;
```

```
  procedure QuickSort(var List: PartNo); external $1C00;
```

Parameter overdracht naar externe subprogramma's wordt verder besproken in 17.14.3
"Parameters".

17.12 Inline machinecode

Turbo Pascal heeft het inline commando om machinecode op te kunnen nemen in de
programma tekst. Een inline opdracht bestaat uit het gereserveerde woord inline gevolgd
door 1 of meer code elementen gescheiden door deelstrepen (/) en omgeven door ronde
haken.

Een code element is opgebouwd uit 1 of meer data elementen, gescheiden door plus (+)
of min (-) tekens. Een data element is of een Integer constante, een naam van een
variabele, een naam van een procedure of functie of een locatie teller verwijzing. Een
locatie teller verwijzing wordt geschreven als een asterisk (*).

Voorbeeld:

```
  inline(10/$2345/Count+1/Sort-*+2);
```

Ieder code element genereert 1 byte of 1 woord (twee bytes) code. De waarde van de byte
(of het woord) wordt berekend door optellen of aftrekken van de data elementen volgens
de tekens die hen scheiden. De waarde van de naam van een variabele/procedure/functie
is het adres (of de offset) van die variabele/procedure/functie. De waarde van de locatie
teller verwijzing is het adres van de volgende te genereren byte van de code.

Een code element genereert 1 byte code als het alleen bestaat uit een Integer constante, en
als zijn waarde binnen de 8-bit grenzen valt (0..255). Als de waarde buiten deze grenzen
valt of als het code element naar een variabele/procedure/functie of locatie teller
refereert, wordt een woord (twee bytes) gegenereerd. De minst belangrijke byte (LSB)

wordt het eerst geschreven.

De '<' en '>' karakters kunnen gebruikt worden om de automatische grootte selectie, die hierboven beschreven wordt, te omzeilen. Als een code element start met een '<' karakter, wordt alleen de minst belangrijke byte van de waarde gecodeerd, zelfs als het om een 16-bits waarde gaat. Als een code element begint met een '>' karakter, wordt er altijd een woord van gemaakt, zelfs als het belangrijkste byte gelijk is aan nul.

Voorbeeld:

```
  inline(<$1234/>$44);
```

Deze inline opdracht levert de volgende drie bytes code op: $34, $44, $00.

Het volgende voorbeeld van een inline opdracht, genereert machinecode die alle karakters van zijn variabele String parameter converteerd naar hoofdletters.

```
procedure UpperCase(var Strg: Str); {Str is van type
                                     String[255] }
{$A+}
begin
  inline ($2A/Strg/        {       LD   HL,(Strg)   }
          $46/             {       LD   B,(HL)      }
          $04/             {       INC  B           }
          $05/             { L1:   DEC  B           }
          $CA/*+20/        {       JP   Z,L2        }
          $23/             {       INC  HL          }
          $7E/             {       LD   A,(HL)      }
          $FE/$61/         {       CP   'a'         }
```

```
          $DA/*-9/          {       JP   C,L1         }

          $FE/$7B/          {       CP   'z'+1        }

          $D2/*-14/         {       JP   NC,L1        }

          $D6/$20/          {       SUB  20H          }

          $77/              {       LD   (HL),A       }

          $C3/*-20);        {       JP   L1           }

                            { L2:   EQU  $            }

end;
```

Inline opdrachten mogen vrij gemengd worden met andere opdrachten door het opdracht-
deel van het blok. Inline statements mogen alle registers van de CPU gebruiken. Merk
echter op dat de inhoud van de stackpointer (SP) bij terugkeer naar gewoon Pascal
hetzelfde moet zijn als voor de uitvoer van de inline.

In Turbo Pascal versie 3.3 is het mogelijk om een procedure of functie direct te laten
beginnen met een inline statement. Men moet dan wel de eventuele parameters zelf van
de stack verwijderen.

Voorbeeld:

```
procedure UpperCase(var Strg: Str); {Str is van type

                                     String[255] }

  inline ($D1/              {       POP DE  ; terugkeer adres}

          $E1/              {       POP HL  ; Strg   }

          $D5/              {       PUSH DE ; herstel}

          $46/              {       LD   B,(HL)       }

          $04/              {       INC  B            }
```

```
$05/            { L1:   DEC  B            }
$CA/*+20/       {       JP   Z,L2         }
$23/            {       INC  HL           }
$7E/            {       LD   A,(HL)       }
$FE/$61/        {       CP   'a'          }
$DA/*-9/        {       JP   C,L1         }
$FE/$7B/        {       CP   'z'+1        }
$D2/*-14/       {       JP   NC,L1        }
$D6/$20/        {       SUB  20H          }
$77/            {       LD   (HL),A       }
$C3/*-20        {       JP   L1           }
                { L2:   EQU  $            }
$C9);           {       RET               }
```

17.13 Interrupt afhandeling

Het Turbo Pascal looptijd (runtime) pakket en de code gegenereerd door de compiler zijn
beide volledig interumpeerbaar. Interrupt routines moeten zelf alle registers herstellen.

Als het nodig is kunnen interrupt routines in Pascal geschreven worden. Zulke procedures
moeten altijd gecompileerd worden met de A directive op actief ({$A+}), ze mogen geen
parameters hebben en ze moeten zelf zorgen dat de gebruikte registers niet wijzigen. Dit
wordt gedaan door het plaatsen van een inline opdracht met de noodzakelijke PUSH
instructies helemaal aan het begin van de routine en een andere inline opdracht met de
corresponderende POP instructies aan het einde van de procedure. De laatste instructie
van deze laatste inline moet zou een EI (Enable Interrupt) moeten zijn (opcode $FB) om
verdere interrupts mogelijk te maken. Als zogenaamde 'daisy-interrupts' worden gebruikt,
kan de inline opdracht ook een RETI (return from interrupt) instructie bevatten (opcode
$ED/$4D), die de RET instructie die door de compiler wordt gegenereerd passeert.

De algemene regels voor het gebruik van de registers zijn dat integer operaties alleen de registers AF, BC, DE en HL gebruiken, andere operaties mogen ook IX en IY gebruiken, real operaties gebruiken de alternatieve registers.

Een interrupt routine mag geen I/O operaties uitvoeren die gebruik maken van de standaard procedures en functies van Turbo Pascal, omdat deze routines niet re-entrant zijn. Merk ook op dat BDOS aanroepen (en in sommige gevallen ook de BIOS aanroepen, afhankelijk van de specifieke CP/M implementatie) niet uitgevoerd mogen worden vanuit interrupt routines, omdat ook deze niet re-entrant zijn.

De programmeur mag de interrupts tijdens de loop van het programma aan- en uitzetten gebruik makend van de EI en DI instructies in inline opdrachten.

Als mode 0 (IM 0) of mode 1 (IM 1) interrupts gebruikt worden, is het de verantwoordelijkheid van de programmeur om de restart locaties te initialiseren op de basis-pagina. (Merk op dat RST 0 niet gebruikt kan worden, omdat CP/M de locaties 0 tot 7 gebruikt).

Als mode 2 (IM 2) interupts worden gebruikt, moet de programmeur een geÔnitialiseerde sprong-tabel genereren (een array van integers) op een absolute adres en het I register initialiseren met een inline statement aan het begin van het programma.

17.14 Interne Data Formaten

In de volgende beschrijvingen, geeft het symbool @ het adres van de eerste bezette byte aan van een variabele van het gegeven type. De standaard functie Addr kan gebruikt worden om deze waarde te verkrijgen voor elke variabele.

17.14.1 Basis Data Typen

De basis data typen kunnen gegroepeerd in structuren (arrays, records en bestanden), maar deze structurering heeft geen invloed op hun interne formaat.

17.14.1.1 Scalairen

De volgende scalairen worden bewaard in een enkele byte: Integer sub-bereiken met

beide grenzen in het bereik 0..255, Booleans, Chars en gedeclareerde scalairen met
minder dan 256 mogelijke waarden. Dit byte bevat de ordinale waarde van de variabele.

De volgende scalairen worden allemaal bewaard in twee bytes: Integers, Integer sub-
bereiken met ÈÈn of beide grenzen buiten het bereik 0..255 en gedeclareerde scalairen met
meer dat 256 mogelijke waarden. Deze bytes bevatten een 2-complements 16-bit waarde
waarvan de minst belangrijke byte (LSB) op het laagste adres staat.

17.14.1.2 Reals

Reals bezetten 6 bytes en ze geven daarmee een 'floating point' getal weer met een 40-bits
mantisse en een 8-bits 2-complements exponent. De exponent wordt bewaard in het
eerste byte en de mantisse wordt bewaard in de volgende 5 bytes met de minst belangrijke
byte het eerst:

@   Exponent

@+1   LSB (minst belangrijke byte) van de mantisse

.

.

@+5   MSB (meest belangrijke byte) van de mantisse

De exponent is in binair formaat en heeft een offset van $80. Dus een exponent van $84
geeft aan dat de mantisse vermenigvuldigd moet worden met 2^($84-$80) = 2^4 = 16.
Als de exponent nul is, wordt de 'floating-point' waarde nul verondersteld.

De waarde van de mantisse wordt verkregen door de 40-bits integer zonder teken te delen
door 2^40. De mantisse is altijd genormaliseerd (de belangrijkste bit (bit 7 van byte @+5)
moet als een 1 geÔnterpreteerd worden. Het teken van de mantisse wordt in dit bit
bewaard, een 1 geeft aan dat de waarde negatief is en een 0 geeft aan dat het om een
positief getal gaat.

17.14.1.3 LongInt

De Longint bezet 4 bytes, waarvan het minst belangrijke byte op het laagste adres staat.

@	Low-Word, low byte (LSB)
@+1	Low-Word, high byte
@+2	High-Word, low byte
@+3	High-Word, high byte (MSB)

17.14.1.4 Strings

Strings bezetten een geheugenruimte van 1 plus de maximale lengte van die String. De eerste byte bevat de lengte van de String. De volgende bytes bevatten de actuele letters, met de eerste letter opgeslagen op het laagste adres. In de tabel hieronder, geeft L de huidige lengte van de String aan en Max geeft zijn maximale lengte aan:

@	Huidige lengte (L)
@+1	Eerste karakter
@+2	Tweede karakter
:	
@+L	Laatste karakter
@+L+1	Ongebruikt
:	
@+Max	Ongebruikt

17.14.1.5 Sets

Een element van een Set heeft 1 bit als opslag en omdat het maximum aantal elementen 256 is, beslaat een Set variabele nooit meer dan 32 bytes (256/8).

Als een Set minder dan 256 elementen bevat zijn enkele van de bits altijd 0 en hoeven dus niet opgeslagen te worden. In termen van geheugen effectiviteit is de beste manier om een Set variabele van een gegeven type op te slaan, het "afkappen" van de onbelangrijke

bits en de rest van de bits zodanig te roteren dat het eerste bit van het eerste byte, het eerste element van de Set zou bevatten. Zulke roteer instructies zijn echter behoorlijk traag en Turbo Pascal werkt daarom met een compromis: Alleen hele bytes die altijd 0 zijn worden niet opgeslagen (bytes waarvan geen enkel bit gebruikt wordt). Deze compressie methode is zeer snel en is in de meeste gevallen even efficiÎnt met het geheugen als de rotatie methode.

Het aantal bytes dat een Set nodig heeft kan berekend worden door (Max div 8) - (Min div 8), waar Max en Min de boven- en ondergrens van het basis type van de Set zijn. Het geheugenadres van een specifiek element E is:

```
  MemAddress = @ + (E div 8) - (Min div 8)
```

en het bit-adres binnen de byte op MemAddress is:

```
  BitAddress = E mod 8
```

waar E de ordinale waarde van het element aangeeft.

17.14.1.6 Bestands variabelen

De onderstaande tabel laat het formaat van een FIB (File Interface Blok) in Turbo Pascal zien:

@+0	Vlaggen byte
@+1	Karakter buffer
@+2	Pointer naar de sector buffer (LSB)
@+3	Pointer naar de sector buffer (MSB)
@+4	Aantal records (LSB)
@+5	Aantal records (MSB)
@+6	Record lengte (LSB)

@+7 Record lengte (MSB)

@+8 Huidig record (LSB)

@+9 Huidig record (MSB)

@+10 Ongebruikt

@+11 Ongebruikt

@+12 Eerste byte van de CP/M FCB

:

@+47 Laatste byte van de CP/M FCB

@+48 Eerste byte van de sector buffer

:

@+175 Laatste byte van de sector buffer

Het formaat van het vlaggen byte is als volgt:

Bit 0..3 Bestand type

Bit 4 Lees semafoor

Bit 5 Schrijf semafoor

Bit 6 Output vlag

Bit 7 Input vlag

Bestand type 0 betekent disk bestand en de typen 1 t/m 5 staan voor de logische I/O apparaten (CON:, KBD:, LST:, AUX: en USR:) van Turbo Pascal. Voor getypeerde bestanden wordt bit 4 gezet als de inhoud van de sector buffer ongedefinieerd is en bit 5 wordt gezet als er data geschreven is naar de sector buffer. Voor tekst bestanden wordt bit 5 gezet als de karakter buffer een vooruit gelezen karakter bevat. Bit 6 wordt gezet als uitvoer toegestaan is en bit 7 wordt gezet als invoer toegestaan is.

De sector buffer pointer bevat een offset (0..127) in de sector buffer op @+48. Voor getypeerde bestanden geven de drie woorden van @+4 tot en met @+9, het aantal

records, de record lengte en het actuele record nummer. Het FIB van een ongetypeerd
bestand heeft geen sector buffer en dus wordt de sector buffer pointer niet gebruikt.

Als een tekst bestand toegewezen wordt aan een logisch apparaat, worden alleen het
vlaggen byte en de karakter buffer gebruikt.

17.14.1.7 Pointers

Een pointer bestaat uit twee bytes die een 16-bit geheugen adres bevatten en het wordt
opgeslagen in het geheugen met het minst belangrijke byte op het laagste adres. De
waarde nil komt overeen met de integer waarde 0.

17.14.2 Data structuren

Data structuren worden opgebouwd uit basis data typen door gebruik van verschillende
structureringstechnieken. Er bestaan drie methoden: arrays, records en disk bestanden.
De structurering heeft geen invloed op het interne data formaat van de basis typen.

17.14.2.1 Arrays

De componenten met de laagste index waarden worden opgeslagen op het laagste adres.
Een meerdimensionale array wordt opgeslagen met een meest rechtse index die als eerste
wordt verhoogd. Bijvoorbeeld als het volgende array gegeven is:

```
  Board: array[1..8,1..8] of Square;
```

dan heb je de volgende geheugen layout van de componenten:

```
laagste adres:        Board[1,1]
           Board[1,2]
           :
           Board[1,8]
           Board[2,1]
           Board[2,2]
```

:

:

Board[8,8]

17.14.2.2 Records

Het eerste veld van een record wordt op het laagste adres opgeslagen. Als het record geen variant gedeelte heeft, wordt de lengte van het record bepaald door de som van de lengten van de individuele velden. Als een record wel een variant gedeelte bevat is het totaal aantal bytes bezet door het record gelijk aan de lengte van het vaste gedeelte plus de lengte van de grootste van zijn variante gedeelten. Elke variant start op hetzelfde geheugenadres.

17.14.2.3 Disk bestanden

Disk bestanden zijn anders dan andere data structuren omdat deze data niet in het interne geheugen wordt opgeslagen, maar in een bestand op een extern apparaat. Een disk bestand wordt bestuurd door een bestand interface blok (FIB) zoals beschreven in 17.14.1.6 "Bestands variabelen". In het algemeen zijn er twee verschillende typen van disk bestanden: random acces bestanden en tekst bestanden.

Random access bestanden

Een random access bestand bestaat uit een opeenvolging van records, allemaal van dezelfde lengte en hetzelfde interne formaat. Om de bestandsopslag capaciteit te optimaliseren, worden records direct aan elkaar geplakt opgeslagen. De eerste vier bytes van de eerste sector van een bestand bevatten, het aantal records in het bestand en de vaste recordlengte van in bytes. Het eerste record van het bestand wordt opgeslagen vanaf het vierde byte.

sector 0, byte 0:	Aantal records (LSB)
sector 0, byte 1:	Aantal records (MSB)
sector 0, byte 2:	Record lengte (LSB)
sector 0, byte 3:	Record lengte (MSB)

Text Bestanden

De basis componenten van een tekst bestand zijn karakters, maar een tekst bestand is ook
onderverdeeld in regels. Elke regel bestaat uit een aantal karakters beÎindigd met een

CR/LF sequentie (ASCII $0D/$0A). Het bestand wordt afgesloten met een Ctrl-Z

(ASCII $1A).

17.14.3 Parameters

Parameters worden aan procedures en functies overgedragen door middel van de CPU
stack. Normaal gesproken is dit niet van belang voor de programmeur, omdat de
machinecode gegenereerd door Turbo Pascal de parameters automatisch op de stack
PUSHt voor de aanroep en ze POPt aan het begin van de procedure of functie. Als de
programmeur echter externe procedures of functies wil gebruiken, moet hij de parameters
zelf van de stack halen.

Bij binnenkomst van een externe subroutine, bevat de top van de stack altijd het terugkeer
adres (ÈÈn woord). De parameters (als die er zijn) staan onder het terugkeer adres (dat wil
zeggen op een hoger adres op de stack). Om de parameters dus te benaderen moet eerst
het terugkeer adres van de stack gehaald worden, daarna alle parameters om daarna het
terugkeer adres weer terug op de stack te zetten.

17.14.3.1 Variabele parameters

Met een variabele (var) parameter, wordt er een woord op de stack geplaatst die het
absolute geheugen adres bevat van de eerste byte van die variabele.

17.14.3.2 Waarde parameters

Bij waarde parameters hangt de overgedragen data af van het type van de parameter zoals
beschreven wordt in de volgende passages.

Scalairen

Integer, Booleans en Chars en gedeclareerde scalairen worden doorgegeven aan de stack
als ÈÈn woord. Als de variabele maar 1 byte beslaat, wordt de belangrijkste byte nul
gemaakt. Normaal gesproken wordt dit woord van de stack afgehaald door een instructie
als POP HL.

LongInt

De Longint bestaat uit vier bytes. Normaal gesproken worden deze bytes van de stack
gehaald door middel van de instructies:

```
  POP HL
  POP DE
```

Van minst belangrijk naar belangrijk zijn de registers dan als volgt te interpreteren:
LHED.

Reals

Een Real wordt naar de stack overgedragen door zes bytes. Als de bytes van de stack
afgehaald worden gebeurt dit met behulp van de volgende instructies:

```
  POP HL
  POP DE
  POP BC
```

dan bevat L de exponent, H het vijfde (minst belangrijke byte), E het vierde, D het derde,
C het tweede en B het eerste (meest belangrijke) byte.

Strings

Als er een String op de top van de stack staat, is het byte waar het register SP naar wijst
de lengte van de String. De bytes op adres SP+1 t/m SP+N (waar N voor de lengte van de
String staat) bevatten de String met het eerste karakter op het laagste adres. De volgende
machinecode instructies kunnen gebruikt worden om een String van de stack te halen en
op te slaan in StrBuf.

```
  LD   DE,StrBuf
  LD   HL,0
  LD   B,H
```

```
ADD  HL,SP
LD   C,(HL)
INC  BC
LDIR
LD   SP,HL
```

Sets

Een Set beslaat altijd 32 bytes op de stack (Set compressie wordt alleen gebruikt bij het laden en het opslaan van Sets). De volgende machinecode instructies kunnen gebruikt worden om een Set van de stack te halen en op te slaan in SetBuf:

```
LD   DE,SetBuf
LD   HL,0
ADD  HL,SP
LD   BC,32
LDIR
LD   SP,HL
```

Dit slaat de minst belangrijke byte van de Set om het laagste adres van SetBuf op.

Pointers

Een pointer wordt op de stack overgedragen als een woord, dat het geheugenadres bevat van een dynamische variabele. De waarde nil komt overeen met een 0 woord.

Arrays en Records

Zelfs als ze gebruikt worden als waarde parameters worden array en record variabelen niet echt op de stack gezet. In plaats daarvan wordt een woord met het adres van het eerste byte van de parameter op de stack gezet. Het is dan de verantwoordelijkheid van de subroutine om dit woord te POPpen en het te gebruiken als het bronadres van een blok copieer instructie.

17.14.4 Functie resultaten

Door de gebruiker gemaakte external functies moeten hun resultaten exact als volgt terug geven:

Waarden van scalaire typen, moeten in het registerpaar HL komen te staan. Als het type uitgedrukt wordt in 1 byte moet het teruggegeven worden in L en H moet dan gelijk aan 0 zijn.

De Longint moet in de registerparen DE en HL terugkomen. Van lage naar hoge belangrijkheid zijn de bytes als volgt te interpreteren LHED (L is het minst belangrijke byte).

Real moeten teruggegeven worden in de registerparen BC, DE en HL. B, C, D, E en H moeten de mantisse bevatten (het belangrijkste byte in B) en L moet de exponent bevatten.

Strings en Sets moeten via de stack teruggegeven worden, zoals beschreven op de vorige pagina.

Pointers moeten teruggegeven worden in het registerpaar HL.

17.14.5 De heap en de stack

Zoals reeds aangegeven door de geheugenkaarten in vorige hoofdstukken, zijn er drie stack-achtige structuren die onderhouden worden tijdens de uitvoer van een programma: de heap, de CPU stack en de recursie stack.

De heap wordt gebruikt om dynamische variabelen op te slaan en kan bestuurd worden door de standaard procedures New, Mark en Release. Aan het begin van een programma wordt de heap pointer HeapPtr op het adres van de bodem van het vrije geheugen gezet (de eerste vrije byte na de object code).

De CPU stack wordt gebruikt om tussenresultaten van expressie evaluaties op te slaan en
tijdens overdracht van parameters naar functies en procedures. Een actieve for opdracht
gebruikt ook de CPU stack en beslaat 1 woord. Aan het begin van een programma wordt
de CPU stackpointer StackPtr op het adres van de top van het vrije geheugen gezet.

De recursie stack wordt alleen gebruikt bij aanroep van recursieve functies en procedures,
(procedures en functies gecompileerd met de A compiler directive op passief ({$A-}).
Bij binnenkomst van het recursieve subprogramma wordt zijn werkgeheugen gecopieerd
naar de recursie stack en bij terugkeer wordt dit werkgeheugen terug gecopieerd, zodat
het weer is zoals het was bij binnenkomst. Bij recursieve overlays worden, per aanroep
naar een overlay, 6 bytes op de heap geplaatst om bij terugkomst de eventueel
overschreven code te kunnen herstellen. De standaard initieele waarde van de
RecurPtr aan het begin van een programma is 1K ($0400 bytes) onder de CPU stackpointer.

Door deze techniek moeten locale variabelen als var parameters doorgegeven worden bij
recursieve aanroepen.

De voorgedefinieerde variabelen:

HeapPtr: De heap pointer

RecurPtr: De recursie stackpointer

StackPtr: De CPU stackpointer

staan het de programmeur toe om de positie van de stacks te manipuleren.

Het type van deze variabelen is Integer. Merk op dat HeapPtr en RecurPtr op
dezelfde manier gebruikt kunnen worden als iedere andere Integer variabele, maar dat de
StackPtr alleen gebruikt mag worden in toewijzingen en in expressies. (Niet als

parameter en dergelijken).

Wanneer deze variabelen verandert worden, moet erop gelet worden dat ze altijd naar vrije geheugen plaatsen wijzen en dat geldt:

```
  HeapPtr < RecurPtr < StackPtr
```

Als men zich niet aan deze regels houd kan dit onvoorspelbare gevolgen hebben, misschien zelfs fatale gevolgen.

Het is onnodig te zeggen, dat toewijzingen aan heap- of stackpointer nooit mogen plaatsvinden als ze in gebruik zijn.

Bij elke aanroep van de procedure New en bij het binnengaan van een recursieve procedure controleert het systeem op "botsing" tussen de heap- en de recursie stack, dat wil zeggen: het controleert of HeapPtr kleiner is dan RecurPtr. Als dat niet zo is dan is er een botsing opgetreden, wat in een looptijdfout resulteert.

Merk op dat er nooit controles worden uitgevoerd om te verzekeren dat de CPU stack niet over de bodem van de recursie stack komt. Om dit te laten gebeuren moet een recursieve routine zich zo'n 300-400 keer aanroepen, wat als een zeldzame gebeurtenis wordt beschouwd. Als een programma dit echter wel nodig heeft, kan de volgende opdracht uitgevoerd worden, die de recursie stack omlaag verplaatst om een grotere CPU stack te creÎren:

```
  RecurPtr := StackPtr - 2*MaxDepth - 512;
```

waar MaxDepth de maximum gewenste diepte van aanroepen naar de recursieve sub-programma's is. De extra 512 bytes zijn nodig voor de opslag van parameteroverdracht en

tussenresultaten tijdens de evaluatie van expressies.

17.14.6 Uitvoer van een programma bestand

Als een programma bestand wordt uitgevoerd, wordt het geheugen als volgt ingedeeld:

Figuur 17 - 1 Geheugen overzicht tijdens de uitvoering van een programma

Het standaard startadres is het eerste vrije byte na de Pascal bibliotheek. Deze waarde kan veranderd worden door het startadres in te stellen d.m.v. de /S compiler optie in de commando regel van Turbo Pascal tijdens de compilatie. De maximum geheugen grootte is BDOS - 1 en de standaard waarde wordt bepaald door de BDOS locatie van de computer die men gebruikt.

Als programma's vertaald worden voor andere systemen, dan moet men oppassen voor een "botsing" met de BDOS. Het maximum geheugen kan beinvloed worden door het eindadres te veranderen.

18. GIOS

Het GIOS is een hulp programma dat vanaf versie 3.3 wordt meegeleverd met Turbo Pascal. GIOS is een afkorting die staat voor Graphical Input Output System. Het is een verzameling procedures en functies die speciaal zijn toegevoegd om in Pascal de grafische mogelijkheden van de MSX ten volle te kunnen gebruiken. Hoewel het GIOS in eerste instantie is opgezet om de grafische mogelijkheden te kunnen benutten, zijn er ook procedures en functies beschikbaar voor de aansturing van diverse rand apparaten zoals muis, joystick, pad enz.

18.1 Opbouw van het GIOS

Het GIOS is opgebouwd uit twee files, namelijk GIOS.COM en GIOS.TSR. Daarnaast is MEMMAN, de geheugenmanager van MST, vereist. De MEMMAN die u gebruikt moet minimaal versie 2.4 zijn en er moet minimaal een heap ruimte beschikbaar zijn van 310 bytes, dit is instelbaar met CFGMMAN.COM.

De TSR is nodig voor het bewaren van het versie nummer van de GIOS, de gegevens
over de twee mapperblokken die de GIOS minimaal gebruikt en de positie van de
gereserveerde heap ruimte. Het versie nummer wordt door Pascal zelf gecontroleerd bij
het uitvoeren van een Pascal programma. Alleen als het versienummer van het GIOS
hoog genoeg is voor deze Pascal versie zal het GIOS worden geactiveerd door het adres
van de MEMMAN heap op te halen. Dit adres wordt opgehaald met behulp van de TSR.
In de MEMMAN heap staat de routine die nodig is om toegang te krijgen tot het GIOS.

GIOS.COM is het deel waarin de uitvoerbare code van alle procedures en functies is
ondergebracht. Door in uw programma een GIOS procedure of functie te gebruiken zal
het geheugenblok waarin dit deel is ondergebracht actief worden, de benodigde
procedure of functie wordt uitgevoerd en er wordt weer teruggekeerd naar uw
programma. Het voordeel van deze methode is dat de grafische routine's niet een enorm
stuk van het beschikbare werkgeheugen gebruiken, dit blijft beschikbaar voor uw eigen
programmacode. Een ander voordeel is alles niet iedere keer opnieuw moeten worden
gecompileerd. Een nadeel van dit systeem is de tijd die verbruikt wordt door het
schakelen van geheugenblokken, net als de BIOS routine's. Overigens is deze tijd te
verwaarlozen als GIOS en Pascal zich in hetzelfde mapper slot bevinden.

Als uw programma bijvoorbeeld echt flitsend snel lijnen moet kunnen tekenen, kunt u
beter gebruik maken van een inline statement die dit verzorgd. Voor normale
toepassingen zijn de GIOS procedures en functies echter snel genoeg.

Op uw diskette vindt u ook nog COMPRESS.COM. Dit is een programma waarmee u files
kunt comprimeren. In het GIOS zit een decompressie routine waarmee u deze files weer
kunt uitpakken in uw eigen programma. Zie hier voor Expand of MemExpand.

COMPRESS.COM is speciaal gemaakt om grafische afbeeldingen in het COPY-formaat te
comprimeren. Om het weergeven van een plaatje zo snel mogelijk te laten verlopen is er

voor gekozen om op byte niveau te werken. Dit houdt in dat op scherm 5 en 7 uw plaatje
op een even X-coˆrdinaat moet beginnen en op een oneven X-coˆrdinaat moet eindigen.
Op scherm 6 moet dit zelfs een veelvoud van vier zijn.

De plaatjes kunnen in BASIC met het COPY-commando worden gemaakt of in Pascal
met SavePicture of MemSavePicture.

18.2 Installatie foutmeldingen

Op het moment dat u een Pascal programma start, wordt de runtime bibliotheek
geinitialiseerd. Tijdens deze installatie wordt gecontroleerd of MEMMAN.COM,
GIOS.TSR en GIOS.COM geladen zijn. Als ÈÈn van deze programma's niet geladen is,
of een verkeerd versie nummer hebben, start uw programma wÈl normaal op. Pas op het
moment dat er gebruik wordt gemaakt van een GIOS procedure of functie en ÈÈn van
deze onderdelen is niet goed geÔnstalleerd dan wordt het programma afgebroken met een
runtime error.

De volgende foutmeldingen kunnen hierbij optreden:

1) runtime error $AA, MEMMAN not present

2) runtime error $AB, Wrong MEMMAN version

3) runtime error $AC, GIOS TSR not present

4) runtime error $AD, GIOS not present

5) runtime error $AE, Wrong GIOS version

U kunt deze foutmeldingen voorkomen door zelf een controle uit te voeren en daardoor
zelf tijdig actie ondernemen voor de runtime error dat doet.

Hiervoor zijn een aantal standaard variabelen toegevoegd aan het Pascal systeem. Deze
variabelen zijn:

Variabele naam

Omschrijving

MemmanPresent

Boolean. Deze is True is als er een MEMMAN geladen is. Het versie nummer is hierbij niet van belang.

MemmanVersion

Integer. Hiermee kunt u zien welke versie van MEMMAN geÔnstalleerd is. Dit versie nummer wordt weergegeven volgens de MEMMAN specificaties. Voor het GIOS moet u minimaal beschikken over MEMMAN 2.4. Als u de waarde opvraagt moet deze minimaal $0204 zijn.

GiosPresent

Boolean. Deze is True is als het GIOS volledig is geÔnstalleerd (zowel TSR als COM file) en de minimaal vereiste versie nummers aanwezig zijn. Als GiosPresent de waarde False heeft kunt u controleren of de TSR geladen is (zie functie TsrPresent) en u kunt het versie nummer van het GIOS controleren.

GiosVersion

Integer. Hiermee kunt u zien welke versie van het GIOS geÔnstalleerd is. Dit versie nummer moet minimaal $0223 zijn en is 0 als GIOS.COM niet is geladen.

Tabel 18 - 1

18.3 Variabelen en typen

Een aantal procedures en functies uit het GIOS gebruiken variabelen uit het werkgebied van MSX-BASIC. Dit zijn de volgende variabelen:

```
LogOpr:  Byte,    adres $FB02

AtrByt:  Byte,    adres $F3F2

ActPage: Byte,    adres $FAF6

Border:  Byte,    adres $F3EB
```

Deze variabelen worden gebruikt voor het instellen van respectievelijk een logische operatie, een kleur, de actieve schermpagina en de randkleur van het scherm. Het voordeel van deze methode is dat deze gegevens niet als parameter hoeven te worden meegegeven. De adressen waarop deze variabelen worden bijgehouden zijn adressen waar ook MSX-BASIC gebruik van maakt.

In de uitleg bij de diverse procedures en functies staat regelmatig ÈÈn van de typen
Str63 en InfoBlock vermeld. Deze typen zijn niet in Pascal gedefinieerd. U zult
deze dus zelf moeten definiÎren. De reden dat deze variabelen niet voorgedefinieerd zijn
is dat het nu eenvoudig mogelijk is om de gegevens in een eigen structuur op te vragen,
let er wel op dat deze eigen structuur 64 bytes groot is. Als de typen voorgedefinieerd
zouden zijn zou dit niet kunnen.

Als u de gegevens niet in een eigen structuur wilt gebruiken dan kunt u de volgende type
definities in uw programma opnemen:

```
type Str63     = String[63];

     Str255    = String[255];

     InfoBlock = array[0..63] of Byte;
```

18.4 GIOS functies en procedures

18.4.1 GIOS functies

Hieronder volgt een gesorteerde lijst van alle functies in het GIOS met een korte
omschrijving.

18.4.1.1 FindFirst

```
Declaratie: function FindFirst(Path: Str63;

                        var Info: InfoBlock;

                        var Attribute: Byte;

                        var Name: Str63): Boolean;
```

Gebruik: BoolResult:=FindFirst(Path, Info, Attribute, Name);

Variabelen: -

Met deze functie kunt u kijken of een bepaalde file in de opgegeven directory voorkomt.

De variabele Path geeft een zogenaamd masker aan zoals dat ook bij het FILES-
commando in MSX-BASIC of het DIR-commando in DOS wordt opgegeven
(bijvoorbeeld A:\TURBO*.* of A:*.PAS). Deze variabele is van het type String en
met een maximum lengte van 63 karakters.

De variabele Info moet een variabele zijn die precies 64 bytes lang is, het type doet er niet toe. Deze variabele wordt gevuld met de informatie die nodig is voor het zoeken van de volgende file (zie FindNext). Onder DOS 1.x wordt deze variabele gevuld met een file control block (FCB), onder DOS 2.x wordt deze variabele gevuld met het info block.

De variabele Attribute wordt als ingaande variabele en als uitgaande variabele gebruikt. In de variabele Attribute wordt aangegeven of files die een bepaalde eigenschap hebben ook moeten worden meegenomen naast de normale files.

Dit attribuut byte is als volgt opgebouwd:

Bit	Waarde	Betekenis
0	+ 1	read-only file
1	+ 2	hidden file
2	+ 4	system file
3	+ 8	volume label
4	+ 16	

subdirectory

5

+ 32

archive bit

6

+ 64

niet in gebruik

7

+ 128

niet in gebruik

Tabel 18 - 2

Als FindFirst True als resultaat heeft, dan zijn de variabelen Info, Attribute en
Name correct ingevuld. In de variabele Attribute staat het bijbehorende attribuut van
deze file, bijvoorbeeld bit 4 is ge-set als het een directory was.

De variabele Info is ingevuld met de informatie voor de eventuele volgende aanroepen
naar de functie FindNext.

In de variabele Name komt de volledige naam van de gevonden file te staan. Deze
variabele moet van het type Str63 zijn omdat onder DOS 2.x een complete file naam
maximaal 63 karakters groot kan zijn.

Deze functie komt volledig overeen met de gelijknamige DOS functie.

Eventuele foutmeldingen zijn op te vragen via de functie GetError.

Zie ook: FindNext, GetError

18.4.1.2 FindNext

```
Declaratie: function FindNext(var Info: InfoBlock;
                              var Attribute: Byte;
```

var Name: Str63): Boolean;

Gebruik: BoolResult:=FindNext(Info, Attribute, Name);

Variabelen:-

Deze functie zoekt naar de volgende file naam vanaf de positie waar een vorige aanroep
van FindFirst of FindNext met hetzelfde InfoBlock gebleven is en geeft een
True als resultaat terug als er een file naam gevonden is die aan de gestelde selectie van
de FindNext voldoet.

In de variabele Attribute komt de informatie te staan over wat voor type file naam het
gaat, zie de functie FindFirst.

In de variabele Name komt de volledige naam van de gevonden file te staan. Deze
variabele moet van het type Str63 zijn omdat onder DOS 2.x een complete file naam
maximaal 63 karakters groot kan zijn.

Onder DOS 2.x wordt alle benodigde informatie om verder te kunnen zoeken uit het
opgegeven InfoBlock gehaald. Dit maakt het mogelijk om meerdere FindFirst en
FindNext aanroepen door elkaar te maken met ieder hun eigen InfoBlock. B.v. het
recursief doorlopen van de directory-structuur.

Onder DOS 1.x wordt er intern een positie bij gehouden waar men gebleven is, de
informatie in het InfoBlock is hier dus alleen een copie van deze interne informatie.
Het gevolg hiervan is dat men onder DOS 1.x de aanroep van FindFirst t/m de laatste
aanroep van FindNext in ÈÈn keer moet uitvoeren.

Zie ook: FindFirst, GetError

18.4.1.3 GetChannel

Declaratie: function GetChannel(Channel: Byte): Real;

Gebruik: RealResult:=GetChannel(Channel);

Variabelen: -

Haalt de positie op van het opgegeven kanaal in Channel. Deze positie varieert van 0
tot en met (Setmem(0) * 16384)-1. Het kanaal nummer mag variÎren van 0 t/m
15. Als het kanaal nummer ongeldig is of het kanaal stond op een ongeldige positie dan
wordt er een -1.0 teruggeven. Ongeldige positie's komen voor als men een positie
opgeeft die groter is of gelijk aan het aantal gereserveerde pagina's * 16384.

In een toekomstige versie van het GIOS zal het resultaat van deze functie veranderen in
een Longint. U kunt nu al zonder problemen direct de waarde toekennen aan een Longint.

Zie ook: SetChannel, SetMem, ReadMem, WriteMem,

MemReadFile, MemWriteFile, MemAppendFile,

MemLoadPicture, MemSavePicture, MemExpand

18.4.1.4 GetClipping

Declaratie: function GetClipping: Boolean;

Gebruik: BoolResult:=GetClipping;

Variabelen: -

Met deze functie kunt u opvragen of op dit moment de clipping geactiveerd is.

Zie ook: SetClipping, SetViewPort, GetViewPort

18.4.1.5 GetDrive

Declaratie: function GetDrive: Integer;

Gebruik: IntResult:=GetDrive;

Variabelen: -

Deze functie haalt het default ofwel het huidige drive nummer op. Deze is 1 voor drive A,
2 voor drive B etc.

Zie ook: TestDrive

18.4.1.6 GetError

Declaratie: function GetError: Byte;

Gebruik: ByteResult:=GetError;

Variabelen: -

Haalt het nummer van de laatst gemaakte foutmelding op. Dit nummer is altijd volgens
de DOS 2 standaard. Onder DOS 1 wordt hiervoor een conversie gemaakt naar de
bijbehorende DOS 2 foutmelding. Alle GIOS procedures en functies die gebruik maken
van de disk kunnen via deze functie een foutmelding terug geven.

Als er geen fout is opgetreden levert deze functie de waarde 0 op. Door de waarde van
deze functie op te vragen wordt de waarde teruggezet naar 0. U kunt de waarde daarom
maar ÈÈn keer opvragen.

Het is zeer raadzaam om, als men foutmeldingen goed wil afvangen, deze functie na
iedere disk benadering aan te roepen en te controleren.

De waarde van IOResult wordt na het lezen van GetError op nul gezet.

Zie de DOS 2 handleiding voor een uitgebreide lijst van foutmeldingen.

18.4.1.7 GetFKey

Declaratie: function GetFKey: Integer;

Gebruik: IntResult:=GetFKey;

Variabelen: -

Deze functie geeft aan welke functietoets is ingedrukt.

Als resultaat wordt 1 t/m 10 teruggegeven voor respectievelijk functietoets 1 t/m 10. Er

wordt een 0 teruggegeven als er geen functietoets is ingedrukt. Als er meerdere functie toetsen worden ingedrukt krijgt u als resultaat het laagste nummer terug.

18.4.1.8 GetPad

Declaratie: function GetPad(Number: Integer): Integer;

Gebruik: IntResult:=GetPad(Number);

Variabelen: -

Met GetPad is het mogelijk om de status van een touchpad, lichtpen, muis of trackball op te vragen.

Voor Number kan men de volgende waarden invullen:

Number	Apparaat / Device
0 t/m 3	Touchpad op joystick poort 1
4 t/m 7	Touchpad op joystick poort 2
8 t/m 11	Lichtpen
12 t/m 15	Muis of trackball op joystick poort 1
16 t/m 19	Muis of trackball op joystick poort 2

Tabel 18 - 3

In het hieronder staand voorbeeld programma wordt alleen het gebruik van GetPad gegeven voor een muis of trackball. Voor documentatie over het gebruik van de touchpad of lichtpen wordt verwezen naar de beschrijving van het PAD-commando in het MSX-BASIC handboek.

Zie ook: GetPdl

18.4.1.9 GetPageID

Declaratie: function GetPageID(Channel: Byte): Integer;

Gebruik: IntResult:=GetPageID(Channel);

Variabelen: -

Haalt de ID-code van de pagina op waar het opgegeven kanaal Channel naar wijst.
Deze ID-code is de 16 bits code die volgens MemMan 2.4 is gespecificeerd. Het kanaal
nummer mag variÎren van 0 t/m 15. Als het kanaal nummer ongeldig is of het kanaal
staat op een ongeldige positie dan wordt er een -1 teruggegeven.

Deze informatie is alleen nuttig te gebruiken voor zeer ervaren programmeurs.

Zie ook: GetChannel, SetChannel

18.4.1.10 GetPdl

Declaratie: function GetPdl(Num: Integer): Integer;

Gebruik: IntResult:=GetPdl(Number);

Variabelen: -

Deze functie leest de waarde van elke paddle die aan een joystick aansluiting is
aangesloten. Aan elk van de zes input lijnen (vier voor de richting en twee voor de
vuurknoppen) van elke aansluiting kan een paddle aangesloten worden, zodat er in totaal
twaalf mogelijk zijn.

Number mag variÎren van 1 t/m 12 waarbij geldt dat de oneven waarden een paddle aan
joystick poort 1 voorstellen en de even waarden een paddle aan joystick poort 2.

Elke paddle is in de grond een monostabiele pulsgenerator, waarvan de pulslengte door
een regelbare weerstand gestuurd wordt. Via register 15 van de PSG wordt een start puls
gestuurd naar de opgegeven input lijn. Vervolgens wordt door de ROM geteld hoe vaak

register 14 van de PSG uitgelezen moet worden voordat het signaal weer '0' wordt, deze
teller is dan het resultaat van deze functie. Met deze functie wordt dus de duur van het
signaal gemeten en daarmee de waarde van de weerstand, dit gebeurd overigens in
eenheden van ongeveer 12 microseconden.

Men kan aan een paddle aansluiting b.v. een potmeter aansluiten en de stand hiervan
uitlezen.

Zie ook: GetPad

18.4.1.11 Point

Declaratie: function Point(X,Y: Integer): Byte;

Gebruik: ByteResult:=Point(X,Y);

Variabelen: ActPage

Deze routine leest het pixel met coˆrdinaat (X,Y) van de actieve pagina ActPage. De
video processor kan het volledige geheugen adresseren met behulp van coˆrdinaten. De Y
waarde kan daarom variÎren van 0 t/m 255. En NIET alleen maar van 0 t/m 211 zoals
MSX2-BASIC doet geloven.

De videoprocessor maskeert zelf de ingevulde (X,Y) coˆrdinaat. De X wordt in scherm 6
en 7 gemaskeerd d.m.v. van 'X and 511'. In alle andere schermen is dit 'X and 255'.
De Y zal altijd begrenst worden op 255. Om een Y waarde van 258 te kunnen
beschrijven moet men eigenlijk kiezen voor ActPage = 1 en Y = 2.

Zie ook: PSet

18.4.1.12 ReadPSG

Declaratie: function ReadPSG(Register: Integer): Byte;

Gebruik: ByteResult:=ReadPsg(Register);

Variabelen: -

Leest de inhoud van het opgegeven registernummer van de PSG. Het registernummer
mag variÎren van 0 t/m 15. De registers 14 en 15 zijn geen registers die te maken hebben
met geluid maar worden gebruikt om de status op te vragen van de twee verschillende
joystick poorten.

Voor details over de inhoud van de registers zie het MSX-BASIC handboek.

Zie ook: WritePSG

18.4.1.13 ReadStatus

Declaratie: function ReadStatus(StatusRegister: Byte): Byte;

Gebruik: ByteResult:=ReadStatus(Register);

Variabelen: -

Leest de inhoud van een status register Register van de VDP. Het statusregister mag
variÎren van 0 t/m 9. Een hogere waarde dan 9 geeft een niet gedefinieerde waarde terug.
Voor de volledigheid staat hieronder een tabel met de overeenkomsten tussen GIOS en
BASIC:

GIOS ReadStatus

BASIC equivalent

0

VDP(8)

1

VDP(-1)

2

VDP(-2)

3

VDP(-3)

4

VDP(-4)

5

VDP(-5)

6

VDP(-6)

7

VDP(-7)

8

VDP(-8)

9

VDP(-9)

Tabel 18 - 4

De aangegeven register waarden onder 'GIOS ReadStatus' zijn de werkelijke status
register nummers zoals de V9938/V9958 die kent.

Voor details over het gedrag van een gelezen status register kunt u de documentatie van
de V9938/V9958 raadplegen.

Zie ook: ReadVDP

18.4.1.14 ReadVDP

Declaratie: function ReadVDP(VDPRegister: Integer): Byte;

Gebruik: ByteResult:=ReadVDP(Register);

Variabelen:-

Leest de inhoud van een VDP register Register terug. Het register nummer mag
variÎren van 0 t/m 23 of 25 t/m 27. Alle andere waarden resulteren altijd in een 0 als
resultaat.

Omdat de VDP geen mogelijkheid biedt om de inhoud van een register terug te lezen

wordt deze bijgehouden in het werkgebied van de MSX. Hiervoor worden de registers op hetzelfde adres bewaard als in MSX-BASIC. Deze adressen zijn:

Register	Adres
0 - 7	F3DF - F3E6
8 - 23	FFE7 - FFF6
25 - 27	FFFA - FFFC

Tabel 18 - 5

Meer details over de VDP registers kunt u vinden in de documentatie van de V9938/ V9958.

Zie ook: ReadStatus

18.4.1.15 Search

Declaratie: function Search(X: Integer; Y: Byte;

Color, Condition: Byte): Integer;

Gebruik: IntegerResult:=Search(X, Y, Color, Condition);

Variabelen: ActPage

Deze functie zoekt vanaf het opgegeven co^rdinaat (X,Y) naar pixels die hetzij gelijk zijn aan de opgegeven kleur Color of die ongelijk zijn aan de opgegeven kleur. Het zoeken gebeurd alleen op een horizontale lijn.

De opgegeven parameter Condition geeft aan hoe er gezocht wordt:

0 = zoek naar rechts totdat de kleur is gevonden.

1 = zoek naar rechts totdat een andere kleur is gevonden.

2 = zoek naar links totdat de kleur is gevonden.

3 = zoek naar links totdat een andere kleur is gevonden.

Het resultaat van deze functie is de X-coˆrdinaat van de pixel die het eerste voldoet aan
de opgegeven conditie. Als de conditie zich niet voordoet voordat het einde van het
scherm wordt bereikt of als de clipping aan staat, voor het einde van de viewport dan
wordt als resultaat de X-coˆrdinaat van de rechter/linker kant van het scherm/viewport
als resultaat teruggegeven. Om een verschil aan te geven tussen een gevonden conditie en
een niet gevonden conditie wordt er $8000 bij de niet gevonden conditie opgeteld. In
praktijk betekend een gevonden waarde dus een positief resultaat en een niet gevonden
een negatief resultaat.

Als de clipping aanstaat en het opgegeven coˆrdinaat (X,Y) valt buiten het viewport, dan
wordt er wel gezocht naar de opgegeven conditie en als normaal een resultaat
teruggegeven. Er wordt dus niet gecontroleerd of het begin punt ook binnen de viewport
valt.

Deze routine is erg snel omdat het hier om een hardwarematige mogelijkheid van de
videoprocessor gaat. Het zoekwerk wordt dus door de VDP uitgevoerd.

Zie ook: Point

18.4.1.16 SetDate

Declaratie: function SetDate(Year: Integer;

Month, Day: Byte): Boolean;

Gebruik: BoolResult:=SetDate(Year, Month, Day);

Variabelen: -

Met deze functie kunt u de datum in de klokchip van de MSX veranderen. De klokchip

accepteert alleen geldige waarden. Als de opgegeven waarden geldig zijn zal deze functie
de waarde True teruggeven. Als de functie False opleverd is de waarde in de klokchip
onveranderd.

Year mag variÎren van 1980 t/m 2079, Month varieert van 1 t/m 12 en Day mag
variÎren van 1 t/m 31.

Zie ook: Date

18.4.1.17 SetMem

Declaratie: function SetMem(Pages: Integer): Integer;

Gebruik: IntResult:=SetMem(Pages);

Variabelen: -

Deze functie reserveert via MemMan een aantal geheugen pagina's die worden
toegevoegd aan de al aanwezige lijst met pagina's. De waarde die wordt teruggegeven
bevat dan ook het aantal al aanwezige pagina's plus de pagina's die met deze functie zijn
aangevraagd en toegewezen. Als het aantal pagina's negatief is of 0 wordt alleen het
aantal al aanwezige pagina's teruggegeven.

Zie ook: ClearMem

18.4.1.18 SetTime

Declaratie: function SetTime(Hour, Minute,Second: Byte): Boolean;

Gebruik: BoolResult:=SetTime(Hour, Minute, Second);

Variabelen: -

Met deze functie kunt u de tijd in de klokchip van de MSX veranderen. De klokchip
accepteert alleen geldige waarden. Als de opgegeven waarden geldig zijn zal deze functie
de waarde True teruggeven. Als de functie False opleverd is de waarde in de klokchip
onveranderd.

Hour mag variÎren van 0 t/m 23, Minute mag variÎren van 0 t/m 59 en Second mag
van 0 t/m 59.

Zie ook: Time

18.4.1.19 SimulatedDisk

Declaratie: function SimulatedDisk(DriveNr: Integer): Boolean;

Gebruik: BoolResult:=SimulatedDisk(DriveNr);

Variabelen: -

Met deze functie kunt u opvragen of een diskdrive wordt gesimuleerd. Als er maar ÈÈn
fysieke diskdrive aanwezig is (harde schijven doen niet mee), dan zal er door de DISK-
ROM altijd een tweede drive gemaakt worden om b.v. het copiÎren van diskette naar
diskette mogelijk te maken.

Als de op dat moment niet actieve diskdrive aangesproken wordt, wordt door de DISK-
ROM de volgende (bijna) niet te vermijden melding gegeven:

'Insert diskette for drive B:'

'and strike a key when ready'

Op grafische schermen resulteerde dit meestal in het blindelings indrukken van een extra
toets, omdat deze melding dan niet te lezen was.

Deze functie probeert de opgegeven drive te benaderen en kijkt of er een melding wordt
gegeven voor een diskwisseling. De melding wordt niet afgedrukt en er hoeft ook geen
toets te worden ingedrukt. Maar er wordt wel als resultaat een True teruggegeven als de
melding is afgedrukt.

Voor DriveNr is 0 voor drive A, 1 voor drive B etc.

Deze functie kan het best gebruikt worden voor dat men b.v. een bestand opent, creÎert of iedere andere functie of procedure die de diskdrive benaderd en waarbij men van te voren niet zeker weet of er een diskwisseling zal plaats vinden. De functie SimulatedDisk benaderd dan zelf eerst de diskdrive en `vangt' eventuele meldingen af.

18.4.1.20 Stick

Declaratie: function Stick(Number: Byte): Byte;

Gebruik: ByteResult:=Stick(Number);

Variabelen: -

Deze functie leest de status van een joystick of de status van de cursor toetsen. Het resultaat hiervan is altijd een waarde die varieert van 0 t/m 8.

Voor Number zijn de volgende mogelijkheden:

Number

Apparaat

0

Toetsenbord

1

Joystick poort 1

2

Joystick poort 2

Tabel 18 - 6

Voor alle andere waarden is de teruggegeven waarde niet gedefinieerd.

Resultaat

Richting

0

geen

1

boven

2

rechtsboven

3

rechts

4

rechtsonder

5

onder

6

linksonder

7

links

8

linksboven

Tabel 18 - 7

18.4.1.21 Strig

Declaratie: function Strig(Number: Byte): Boolean;

Gebruik: BoolResult:=Strig(Number);

Variabelen: -

Met deze functie kunt u opvragen of de spatiebalk of ÈÈn van de 4 mogelijke
vuurknoppen is ingedrukt.

Number

Vuurknop

0

spatiebalk

1

vuurknop 1, joystick poort 1

2

vuurknop 1, joystick poort 2

3

vuurknop 2, joystick poort 1

4

vuurknop 2, joystick poort 2

Tabel 18 - 8

Voor alle overige waarden van Number is de uitkomst van het resultaat ongedefinieerd.

18.4.1.22 TestDrive

Declaratie: function TestDrive(DriveNumber: Byte): Boolean;

Gebruik: BoolResult:=TestDrive(Number);

Variabelen: -

Controleert of Number een correcte drive is. Als Number correct is dan wordt een True
teruggegeven. Number is 1 voor drive A, 2 voor drive B etc..

Het maximale aantal drives in een MSX is 8. Als Number groter of gelijk is aan 9 dan
wordt altijd een False teruggegeven. In het geval van een 0, de default drive, wordt altijd
een True teruggegeven.

Zie ook: GetDrive

18.4.1.23 TsrPresent

Declaratie: function TsrPresent(Name: Str255): Boolean;

Gebruik: BoolResult:=TsrPresent(Name);

Variabelen: -

Met deze functie kunt u controleren of een bepaalde MEMMAN-TSR aanwezig is.

Via Name geeft u de naam op die u wilt controleren. Dit moet op dezelfde manier
gebeuren als via het MEMMAN programma TK.COM. U moet dus vooraf precies weten
welke TSR u gaat controleren. Het is met deze functie niet mogelijk om op te vragen
welke TSR's er aanwezig zijn. U kunt aanwezige TSR's opvragen met het
hulpprogramma TV.COM. De namen die hierin worden getoond hebben precies de goede
schrijfwijze en kunnen direct worden gebruikt in deze functie.

De functie TsrPresent wordt intern in TURBO gebruikt om te controleren of de
GIOS-TSR is geladen. Als deze niet geladen is en uw programma roept een GIOS
procedure of functie aan, dan wordt de runtime error $AC (GIOS.TSR not present)
gegenereerd.

Als u zelf wilt kijken of de GIOS-TSR aanwezig is kunt u dit doen met:

```
  BoolResult:=TsrPresent('FH GIOS 2.1');
```

Let er hierbij op dat de naam van een TSR maximaal 12 karakters lang is volgens de
MemMan specificaties. Als de naam korter is of langer dan wordt deze zonodig
aangevuld met spaties of afgekapt op 12 karakters.

Hoewel deze functie een onderdeel is van het GIOS, is deze ondergebracht in de runtime
bibliotheek van Turbo Pascal. Hierdoor is deze functie ook beschikbaar als het GIOS niet
is geactiveerd.

18.4.1.24 VPeek

Declaratie: function Vpeek(Address: Integer): Byte;

Gebruik: ByteResult:=VPeek(Address);

Variabelen: ActPage

Deze functie leest het opgegeven VRAM adres en resulteert in het gelezen byte. Er wordt gelezen vanaf de pagina die is opgegeven in ActPage. Als ActPage=2 in scherm mode 7 of hoger dan leest men uit het eventueel aanwezige externe VRAM. Als ActPage=4 of ActPage=5 in scherm mode 0 t/m 6 dan leest men in die modes uit het externe VRAM.

Zie ook: VPoke

18.4.2 GIOS procedures

18.4.2.1 BLoad

Declaratie: procedure Bload(Offset: Integer; Page: Byte;

Name: Str63);

Gebruik: Bload(Offset,Page,Name);

Variabelen: -

Deze procedure laad een plaatje in het VRAM op pagina Page en op het adres dat de som is van het beginadres zoals aangegeven in de bload file en de opgegeven Offset.

Er wordt geen controle gedaan of de file ook een echte bload file is.

Om het mogelijk te maken dat men de 128 Kbyte grote files die men met een BSave kan maken ook te kunnen inladen wordt er vanaf het beginadres + Offset net zo lang data geschreven naar het VRAM totdat het einde van de file bereikt wordt.

Alle fouten die ontstaan tijdens het laden kunnen worden opgevraagd via de functie GetError.

Zie ook: BSave

18.4.2.2 BSave

```
Declaratie:procedure BSave(Address1:  Integer; Page1: Byte;
                         Address2: Integer; Page2: Byte;
                         Name: Str63);
```

```
Gebruik:   BSave(Address1,Page1,Address2,Page2,Name);
```

Variabelen:-

Deze procedure creÎert een bload file met de opgegeven naam. Het VRAM gebied dat
wordt weggeschreven is datgene dat zich tussen de opgegeven start en eindadres op de
opgegeven pagina bevindt. Om de pagina en het adres te combineren wordt intern de
volgende formule gebruikt:

scherm mode 0 t/m 6

werkadres = pagina * 32768 + adres

scherm mode 7,8,10,11,12

werkadres = pagina * 65536 + adres

Deze formules leveren een 17 bits getal op.

De weg te schrijven data begint vanaf het laagste werkadres en gaat door t/m het hoogste
werkadres.

De header van een bload file bestaat uit de volgende componenten:

byte $FE, beginadres, eindadres, startadres.

Het beginadres en het startadres worden gevuld met de laatste 16 bits van het laagste
werkadres. Het eindadres wordt gevuld met de laatste 16 bits van het hoogste werkadres.

Met deze procedure is het mogelijk om de complete 128 Kbyte aan VRAM weg te
schrijven. Het zal duidelijk zijn dat dan een eindadres van 16 bits nooit een correct
eindadres zal zijn. Er wordt ook nooit iets gedaan met dit eindadres, zoals vermeld bij de
Bload procedure.

Ook met deze procedure is het mogelijk om het externe VRAM te benaderen op dezelfde
manier als bij de Bload procedure maar dan moeten beide pagina's wel groter of gelijk
zijn aan 4 voor de schermen 0 t/m 6 en groter of gelijk aan 2 voor de schermen 7, 8 en 10
t/m 12. Het is niet mogelijk het normale VRAM en het extended VRAM in ÈÈn file weg
te schrijven.

Zie ook: BLoad

18.4.2.3 ChangeColor

Declaratie: procedure ChangeColor(ColorNr: Byte;

Red, Green, Blue: Byte);

Gebruik: ChangeColor(ColorNr, Red, Green , Blue);

Variabelen: -

Deze procedure wijzigt van het opgegeven kleur nummer ColorNr de R, G, B intensiteit in Red, Green en Blue.

Dit is gelijk aan de werking van het COLOR=(ColorNr, Red, Green, Blue) commando
van BASIC, met dit verschil dat deze procedure geen copie van het complete palet in het
VRAM bijhoudt op een eventuele ongewenste plek.

18.4.2.4 ChangeScreen

Declaratie: procedure ChangeScreen(ScreenNr: Byte);

Gebruik: ChangeScreen(ScreenNr)

Variabelen: -

Deze procedure wijzigt de scherm mode, zonder daarbij het VRAM te beÔnvloeden, in de
opgegeven schermmode ScreenNr. Bij keuze van scherm mode 0 wordt afhankelijk
van de inhoud van systeem adres $F3B0 gekozen voor een 40 koloms mode (inhoud
$F3B0 kleiner dan 41) of anders een 80 koloms mode.

Voor de werkelijke mode wijziging worden alleen de mode bits in register 0 en 1 van de
VDP beschreven en de YAE en YJK bits in register 25.

Het normale Screen commando initialiseerd ook VDP registers voor b.v. de sprites
zodat deze op de standaard adressen komen te staan, maar deze worden dus niet door
deze procedure beÔnvloed.

Net als bij de procedure Screen wordt ook hier de viewport gereset naar de afmetingen
van de opgegeven schermmode.

Zie ook: Screen

18.4.2.5 Circle

Declaratie: procedure Circle(X, Y, Radius: Integer);

Gebruik: Circle(X, Y, Radius);

Variabelen: AtrByt, LogOpr, ActPage

Deze procedure tekent een cirkel op het scherm in de kleur zoals aangegeven in AtrByt
en op de pagina aangegeven in ActPage.

X, Y en Radius worden gegarandeerd binnen een bereik van -10000 tot +10000. Het
Circle commando kan geclipt worden zodat alleen die pixels te voorschijn komen die
binnen de gedefinieerde viewport vallen.

U kunt gebruik maken van logische bewerkingen.

Zie ook: Ellipse

18.4.2.6 ClearMem

Declaratie: procedure ClearMem;

Gebruik: ClearMem;

Variabelen: -

Deze procedure maakt alle geheugen blokken die door de functie SetMem waren
gereserveerd weer vrij voor gebruik.

Als u gebruik maakt van dit extra geheugen dan moet u deze procedure als eerste
aanroepen als u niet zeker weet of dit al gebeurd is.

Waarschuwing: Als u aan het einde van uw programma het gereserveerde geheugen niet
vrijgeeft zal MemMan dit geheugen gereserveerd houden. Indien u dan de GIOS.TSR
verwijderd zal dit geheugen nooit meer vrijkomen voor andere applicaties. U mag het
geheugen daarom alleen vasthouden als u zeker weet dat een vervolg programma dit
geheugen weer vrijgeeft of er van gebruik maakt.

Zie: SetMem

18.4.2.7 Date

Declaratie: procedure Date(var Year: Integer;

var Month, Day, WeekDay: Byte);

Gebruik: Date(Year, Month, Day, WeekDay);

Variabelen: -

Deze procedure haalt de huidige datum op. Year varieert van 1980 t/m 2079, Month
varieert van 1 t/m 12 en Day varieert van 1 t/m 31. Weekday varieert van 0 t/m 6 voor
respectievelijk Zondag, Maandag, Dinsdag, Woensdag, Donderdag, Vrijdag en Zaterdag.

Zie ook: SetDate

18.4.2.8 DefinePicture

Declaratie: procedure DefinePicture(X1, Y1, X2, Y2,

OffsetX, OffsetY: Integer;

Page: Byte);

Gebruik: DefinePicture(X1,Y1,X2,Y2,OffsetX,OffsetY,Page);

Variabelen:-

Deze routine definieert het plaatje zoals dat gebruikt moet worden bij een PFillShape
of een PPaint. Dit plaatje wordt gedefinieerd met behulp van twee coˆrdinaten paren
(X1,Y1) en (X2,Y2) en de pagina waar het plaatje staat. OffsetX en OffsetY geven
hierbij een verschuiving aan.

U kunt zich dit voorstellen als een stempel. Deze stempel wordt gebruikt om een
denkbeeldig scherm helemaal vol te zetten met horizontale- en verticale rijen van het
gekozen plaatje. Als u nu het in te kleuren vlak als een transparant vlak ziet, zult u door
dit transparante vlak heen dit `achterliggende' scherm zien.

Met de OffsetX en OffsetY kunt u dit denkbeeldige scherm eventueel verschuiven.
Als u opgeeft: OffsetX=1 en OffsetY=-2 dan zal het denkbeeldige achterliggende
scherm 1 pixel naar links schuiven en 2 pixels naar beneden.

Zie ook: PFillShape, PPaint

18.4.2.9 DeleteFile

Declaratie: procedure DeleteFile(FileName: Str63);

Gebruik: DeleteFile(FileName);

Variabelen:-

Deze routine wist de opgegeven file FileName. Deze file naam mag een complete
drive/path/name zijn zoals onder DOS 2 gewoon is.

Alle fouten die ontstaan tijdens het wissen kunnen worden opgevraagd via de functie
GetError. Dit geldt dus ook voor fouten als 'disk not ready'.

18.4.2.10 DisplayPage

Declaratie:procedure DisplayPage(PageNr: Byte);

Gebruik: DisplayPage(PageNr);

Variabelen: -

Deze procedure laat de pagina PageNr zien, in de grafische mode. Deze functie werkt
alleen goed in ÈÈn van de grafische modes vanaf scherm 5.

De opdracht 'WriteVDP(2,((PageNr and 3) * 32) or $1f)' doet precies hetzelfde.

Het extended VRAM is nooit zichtbaar te maken en het invullen van pagina's die in het
extended VRAM thuishoren heeft een nogal onvoorspelbaar resultaat.

18.4.2.11 Ellipse

Declaratie: procedure Ellipse(X, Y, RadiusX, RadiusY: Integer);

Gebruik: Ellipse(X,Y,RadiusX,RadiusY);

Variabelen: LogOpr, ActPage, AtrByt

Deze procedure tekent een ellips op het scherm in de kleur zoals aangegeven in AtrByt
en op de pagina aangegeven in ActPage.

De waarden van X en Y mogen ver buiten het scherm vallen zowel negatief als positief.
Afhankelijk van de grootte van de RadiusX en/of RadiusY kan men dan toch nog een
bepaalde sector van de ellips zien.

X, Y, RadiusX en RadiusY worden gegarandeerd binnen een bereik van -10000 tot
+10000.

Het Ellipse commando kan geclipt worden zodat alleen die pixels te voorschijn
komen die binnen de definieerde viewport vallen.

Als bij het Ellipse commando een te grote afplatting wordt gemaakt dan kan het
voorkomen dat twee pixels dubbel worden getekend, dit in tegenstelling tot de procedure
Circle.

In de schermmode 6 en 7 is het Ellipse commando nodig voor het tekenen van een

ronde cirkel. Dit komt omdat de pixel verhouding op deze schermen anders zijn.

Zie ook: Circle

18.4.2.12 Expand

Declaratie: procedure Expand(X,Y: Integer; Page: Byte;

Name: Str63);

of: procedure Expand(X,Y: Integer; Page: Byte;

Name: Str63; Pos: Longint);

Gebruik: Expand(X,Y,Page,Name);

of: Expand(X,Y,Page,Name,Pos);

Variabelen: -

Deze routine expandeert een met het 'compress.com' programma gecomprimeerd
plaatje direct in het VRAM vanaf de opgegeven co^rdinaat (X,Y) in ÈÈn van de vier
mogelijke richtingen. Deze procedure is byte-georiÎnteerd.

Als X en/of Y negatief zijn dan wordt de richting van de Expand gewijzigd in
respectievelijk rechts naar links of van omlaag naar omhoog.

Normaal heeft deze procedure vier parameters maar met behulp van de vijfde parameter
kan men ook de positie binnen de file opgeven van het te expanderen plaatje. Op deze
positie moet dan wel een correcte expand-header staan. In het geval van een incorrecte
header zal via de functie GetError foutcode 63 worden teruggegeven.

Zie ook: MemExpand

18.4.2.13 FastBox

Declaratie: procedure FastBox(X1, Y1, X2, Y2: Integer);

Gebruik: Fastbox(X1,Y1,X2,Y2);

Variabelen: ActPage, AtrByt

Deze procedure tekent een rechthoek die begint op (X1,Y1) en eindigt op (X2,Y2) op de
actieve pagina ActPage. Deze functie is byte-georiÎnteerd en kent daarom geen
logische operaties.

De variabele AtrByt wordt gebruikt bij het bepalen van de kleur. Dit gebeurd op byte
niveau, dit betekend dat op de schermen 5, 6 en 7 er 2 of 4 pixels met ÈÈn byte worden
gevuld. Het is hierdoor mogelijk om b.v. op de schermen 5 en 7 een streepeffect te
krijgen door in de hoge en lage nibble van AtrByt een andere kleur te definiÎren.

U kunt met deze procedure ook in het extended VRAM een box tekenen.

Zie ook: FillBox

18.4.2.14 FastCopy

Declaratie: procedure FastCopy(X1, Y1, X2, Y2,

DX, DY: Integer; DestPage: Byte);

Gebruik: FastCopy(X1,Y1,X2,Y2,DX,DY,DestPage);

Variabelen: ActPage

FastCopy copieert een rechthoek die begint op (X1,Y1) en eindigt op (X2,Y2) naar de
rechthoek die begint op (DX,DY) die zich bevindt op de opgegeven pagina DestPage.
Het origineel staat op de actieve pagina ActPage.

Het copiÎren gebeurt afhankelijk van de ligging van de coˆrdinaten (X1,Y1) en (X2,Y2)
in ÈÈn van de vier mogelijk richtingen. Het copiÎren gebeurt byte-georiÎnteerd hetgeen
een aanzienlijke snelheidsverbetering opleverd t.o.v. GCopy. Er is dan ook geen logische
operatie mogelijk.

Het is mogelijk om met deze procedure van en naar het externe VRAM te copiÎren.

Zie ook: GCopy

18.4.2.15 FillBox

Declaratie: procedure FillBox(X1, Y1, X2, Y2: Integer);

Gebruik: FillBox(X1,Y1,X2,Y2);

Variabelen: AtrByt, ActPage, LogOpr

Deze procedure tekent een rechthoek die begint op (X1,Y1) en eindigt op (X2,Y2) op de actieve pagina ActPage. Deze functie is pixel-georiÎnteerd en kent daarom logische operaties die via de variabele LogOpr kunnen worden opgegeven.

U kunt met deze procedure ook in het extended VRAM een box tekenen.

Zie ook: FastBox

18.4.2.16 FillShape

Declaratie: procedure FillShape(X: Integer; Y, Color: Byte);

Gebruik: FillShape(X,Y,Color);

Variabele: ActPage, Logopr

Deze routine kleurt vanaf het coˆrdinaat (X,Y) alle aanliggende pixels met dezelfde kleur als op het begin coˆrdinaat (X,Y) in met de opgegeven kleur Color. Het resultaat komt op de pagina in ActPage.

Er wordt meteen gestopt als de opgegeven (X,Y) coˆrdinaat buiten het scherm valt of niet binnen de gedefinieerde viewport valt. Ook op deze functie is clipping van toepassing.

Een logische operatie kan worden opgegeven via de variabele LogOpr. U kunt met deze

procedure ook in het extended VRAM een vlak inkleuren.

Zie ook: Paint, PFillShape

18.4.2.17 FillSprite

Declaratie: procedure FillSprite(Plane, Color: Byte);

Gebruik: FillSprite(Plane,Color);

Variabelen: -

Deze procedure geeft het sprite dat zich op 1 van de 32 planes bevind de opgegeven kleur
Color.

Om dit te doen wordt in de spritekleurtabel, die zich 512 bytes lager bevindt dan de
spriteattribuuttabel, een reeks van 8 bytes of 16 bytes gevuld met de opgegeven kleur.

De kleur wordt als byte overgenomen zodat de speciale mogelijkheden die de hoogste 4
bits bieden ook benut kunnen worden.

Zie ook: SpriteColor

18.4.2.18 GCopy

Declaratie: procedure GCopy(X1, Y1, X2, Y2: Integer;

DX, DY: Integer; DestPage: Byte);

Gebruik: GCopy(X1,Y1,X2,Y2,DX,DY,DestPage);

Variabelen: ActPage, LogOpr

GCopy copieert een rechthoek die begint op (X1,Y1) en eindigt op (X2,Y2) naar de
rechthoek die begint op (DX,DY) die zich bevindt op de opgegeven pagina DestPage.
Het origineel staat op de actieve pagina ActPage.

Het copiÎren gebeurt afhankelijk van de ligging van de coˆrdinaten (X1,Y1) en (X2,Y2)

in ÈÈn van de vier mogelijke richtingen. Het copiÎren gebeurt pixel-georiÎnteerd hetgeen het gebruik van een logische operatie mogelijk maakt. De logische operatie kan worden opgegeven via de variabele LogOpr.

Het is mogelijk om met deze procedure van en naar het externe VRAM te copiÎren.

Zie ook: FastCopy

18.4.2.19 GetDosVersion

Declaratie: procedure GetDosVersion(var Kernel, System: Integer);

Gebruik: GetDosVersion(Kernel,System);

Variabelen: -

Met deze procedure is het mogelijk om de versie nummers van de gebruikte DOS versie te bepalen. In Kernel wordt de versie teruggegeven van de gebruikte DOS 2 ROM en in System wordt de versie teruggegeven van de gebruikte `MSXDOS2.SYS' file.

Onder DOS 2.32 zal bijvoorbeeld voor kernel de waarde $0232 worden gegeven en voor System de waarde $0232. Onder DOS 1 zijn beide waarden gelijk aan $0100.

In het algemeen zal men alleen geÔnteresseerd zijn of men werkt onder DOS 1 of DOS 2.

18.4.2.20 GetViewPort

Declaratie: procedure GetViewPort(var X1, Y1, X2, Y2: Integer);

Gebruik: GetViewPort(X1,Y1,X2,Y2);

Variabelen: -

Deze procedure haalt de huidige viewport instelling op. De huidige instelling is de instelling na de laatste Screen procedure, de laatste SetViewPort procedure of de standaard instelling bij het opstarten van de GIOS, namelijk (0,0,255,255).

In het co^rdinaat (X1,Y1) staat de linkerbovenhoek en in (X2,Y2) staat de rechteronderhoek.

Zie ook: SetViewPort, SetClipping, Screen

18.4.2.21 Line

Declaratie: procedure Line(X1, Y1, X2, Y2: Integer);

Gebruik: Line(X1,Y1,X2,Y2);

Variabelen: LogOpr, ActPage, AtrByt

Deze procedure tekent een rechte lijn die begint op (X1,Y1) en eindigt op (X2,Y2) op de
pagina ActPage in de kleur AtrByt. Er kan gebruik gemaakt worden van een logische
operatie LogOpr.

Het tekenen gebeurt afhankelijk van de ligging van de co^rdinaten (X1,Y1) en (X2,Y2) in
ÈÈn van de vier mogelijke richtingen.

Opmerking: Line(0,0, 10,10) is wat het lijn commando betreft hetzelfde als
Line(10,10, 0,0). Maar voor bijvoorbeeld het GCopy commando is dit vaak wel
iets TOTAAL anders, omdat de eerste van links boven naar rechts onder werkt. En de
tweede andersom.

18.4.2.22 LoadPicture

Declaratie: procedure LoadPicture(X: Integer; Y, Page: Byte;

Name: Str63);

Gebruik: LoadPicture(X,Y,Page,Name);

Variabelen: LogOpr

Deze procedure laad een plaatje vanuit de file Name in het VRAM beginnend op de
opgegeven co^rdinaat (X,Y) en op de opgegeven pagina Page. De breedte en hoogte van

het plaatje staan als eerste in de data van het plaatje en hoeven dus niet te worden opgegeven.

Als voortijdig het einde van de file wordt bereikt stop de hele operatie, dit heeft verder geen enkele consequenties. Het is de verantwoordelijkheid van de programmeur om de data correct aan te bieden.

Deze functie zal als het kan byte-georiÎnteerd uitgevoerd worden. Dit hangt af van:

LogOpr (deze moet 0 zijn),

de begin X-coˆrdinaat en

de breedte van het plaatje.

Zie ook: SavePicture

18.4.2.23 LoadVRAM

Declaratie: procedure LoadVRAM(Address: Integer; Page: Byte;

Position,Size: Real; Name: Str63);

Gebruik: LoadVRAM(Address,Page,Position,Size,Name);

Variabelen: -

Deze procedure leest vanuit de opgegeven file Name vanaf de opgeven positie Position een totaal van Size bytes en schrijft deze naar het opgeven VRAM adres Address op de opgegeven pagina Page.

De data die wordt geschreven bestaat uit een reeks bytes zonder enig formaat. U kunt dus een volledig willekeurige file naar het VRAM schrijven. Als voortijdig het einde van de file wordt gevonden, wordt er gestopt. Er wordt dus niet meer data naar het VRAM geschreven als er vanaf de opgegeven positie aanwezig is.

Eventuele disk foutmeldingen kunnen worden opgevraagd via de functie GetError.

18.4.2.24 MemAppendFile

Declaratie: procedure MemAppendFile(Channel: Byte;Position:Real;

Size: Real; Name: Str63);

Gebruik: MemAppendFile(Channel, Position, Size, Name);

Variabelen: -

Deze procedure schrijft data naar de opgegeven file beginnend op de opgegeven file
positie. Deze data heeft een grootte van Size bytes en wordt gelezen vanaf de positie
van het opgegeven kanaal.

Deze procedure overschrijft of voegt data toe aan een al bestaande file. Als de file nog
niet bestond dan wordt deze automatisch aangemaakt.

Eventuele disk foutmeldingen kunnen worden opgevraagd via de functie GetError.

Zie ook: MemReadFile, MemWriteFile

18.4.2.25 MemCopy

Declaratie: procedure MemCopy(Source, Destination: Byte;

Length: Longint);

Gebruik: MemCopy(Source,Destination,Length);

Variabelen: -

Deze procedure copieert data van het Source kanaal naar het Destination kanaal,
hierbij worden maximaal Length bytes gecopieerd.

Het copiÎren gebeurd in oplopende volgorde. Na afloop is het Source kanaal in totaal
Length bytes opgehoogd of ongeldig geworden, hetzelfde geldt voor het Destination kanaal.

Dit commando werkt altijd correct als de positie van het Destination kanaal een
lagere waarde heeft dan die van het Source kanaal. Als dit niet zo is dan kunnen er

'fouten' optreden als beide gebieden een overlapping hebben.

Stel het Source kanaal staat op positie 100 en het Destination kanaal op positie 200
en men gaat een MemCopy uitvoeren met een lengte van 140000. Het volgende gebeurd:

- de eerste 100 bytes worden correct gecopieerd,

- de tweede 100 bytes van het Source kanaal nu zijn overschreven door het
Destination kanaal,

- de tweede 100 bytes worden dan opnieuw gecopieerd naar een positie die 100 bytes
verder staat,

- wat volgt is dat over de volledige lengte van 140000 bytes de eerste 100 bytes van het

Source kanaal vanaf de begin positie van het Destination kanaal 1400 keer herhaald voorkomen.

Stel het verschil tussen het Source kanaal en het Destination kanaal was 1 en het
eerste byte was een 0 dan zou het complete gebied vanaf het Destination kanaal
gevuld worden met 0. Dit kan soms best handig zijn.

18.4.2.26 MemExpand

Declaratie: procedure MemExpand(X, Y: Integer;

Page, Channel: Byte);

Gebruik: MemExpand(X,Y,Page,Channel);

Variabelen: -

Deze routine expandeert een, met het 'compress.com' programma, gecomprimeerd
plaatje. Dit plaatje wordt direct weergegeven op het scherm op de opgegeven (X,Y)
co^rdinaat en de opgegeven pagina Page.

Van de waarde van X en Y wordt de absolute waarde genomen en waarna ze pas worden
gebruikt. Als X en/of Y negatief zijn dan wordt de richting van de MemExpand gewijzigd

in respectievelijk rechts naar links of van omlaag naar omhoog. Omdat de MemExpand
byte georiÎnteerd is, is er een beperking op de X-coˆrdinaten in scherm 5, 6 en 7.

De data moet klaar staan op het opgegeven kanaal. Deze data wordt in deze GIOS
voorafgegaan van een 33 bytes lange 'header'. Als deze header niet correct is dan wordt
via GetError foutmelding 63 (illegal header) teruggegeven. Deze foutmelding valt
binnen het bereik van de zogenaamde 'user errors' van DOS 2. Files die zijn aangemaakt
volgens de oude versie kunnen gewoon aangepast worden voor gebruik met de nieuwe
versie door er deze header voor te plaatsen.

Een gecomprimeerde file heeft de volgende globale indeling:

1. Header.

2. Optioneel gereserveerde data (b.v. paletdata). Dit gedeelte is maximaal 8192
bytes groot.

3. Breedte en Hoogte van het plaatje. Dit is eigenlijk een copie van de eerste 4
bytes van de originele file. Er wordt dus vanuit gegaan dat de originele file b.v.
is gemaakt d.m.v. ÈÈn van de procedures SavePicture, MemSavePicture of het COPY-commando van BASIC.

4. De data van het plaatje zelf.

5. Eventueel een herhaling van de eerste 4 punten voor b.v. een tweede MemExpand.

Punt 3 en 4 samen zijn exact gelijk aan de data die door de oude 'compress.com' werd
afgeleverd.

Hieronder een record dat de inhoud van deze header beschrijft:

```
type ExpandHeader =

        record
```

```
    HeaderName:     array[0..12] of Char;
    ScreenNumber:   Byte;
    DataLength:     Longint;
    ReservedLength: Integer;
    FileName:       array[0..12] of Char;
  end;
```

In de volgende tabel staat hoe deze velden worden ingevuld.

Veld	Inhoud
HeaderName	'GIOS CMP 1.0 ', twaalf tekens waaronder 3 spaties !
ScreenNumber	0 .. 8 en 10 .. 12. Het scherm nummer wordt door de GIOS niet gecontroleerd en kan in principe iedere waarde van 0 t/m 255 bevatten.
DataLength	Dit is de lengte van punt 3 en 4. Deze waarde is minimaal 4.
ReservedLength	Dit is de lengte van punt 2. Deze mag 0 zijn. Expand en MemExpand negeren dit gebied.
FileName	ASCIIZ-string (dit is een reeks karakters afgesloten met karakter 0). Maximaal 12 letters gevolgd door een #0. V.b. 'TEST.GE5'+#0

Tabel 18 - 9

Zie ook: Expand

18.4.2.27 MemLoadPicture

Declaratie: procedure MemLoadPicture(X: Integer; Y: Byte;

Page, Channel: Byte);

Gebruik: MemLoadPicture(X,Y,Page,Channel);

Variabelen: LogOpr

Deze procedure laadt een plaatje vanuit de memory mapper in het VRAM beginnend op
de opgegeven coˆrdinaat (X,Y) en op de opgegeven pagina Page. De breedte en hoogte
van het plaatje staan als eerste in de data van het plaatje en hoeven dus niet te worden
opgegeven. Het plaatje begint op de plek waar het waar opgegeven kanaal Channel
naar wijst.

Het kanaal mag variÎren van 0 t/m 15. Als het kanaal nummer ongeldig is of het kanaal
stond op een ongeldige positie dan wordt deze procedure niet uitgevoerd.

Als er vanaf de positie van het gegeven kanaal niet genoeg ruimte of exact genoeg ruimte
is dan is het kanaal na afloop ongeldig gemaakt. D.w.z. dat een eventuele GetChannel
op hetzelfde kanaal een -1 als resultaat geeft.

Als voortijdig het einde van het kanaal wordt gevonden wordt er gestopt, dit heeft verder
geen enkele consequenties. Het is de verantwoordelijkheid van de programmeur om de
data correct aan te bieden.

Deze functie zal als het kan byte-georiÎnteerd uitgevoerd worden. Dit hangt af van:

LogOpr (deze moet 0 zijn),

de begin X-coˆrdinaat en

de breedte van het plaatje.

Zie ook: MemSavePicture

18.4.2.28 MemReadFile

Declaratie: procedure MemReadFile(Channel: Byte; Position: Real;
Size: Real; Name: Str63);

Gebruik: MemReadFile(Channel,Position,Size,Name);

Variabelen:-

Deze procedure leest uit de opgegeven file beginnend de opgegeven file positie een aantal
bytes ter grootte van Size. Deze bytes worden geschreven vanaf de positie van het
opgegeven kanaal.

Er worden maximaal Size bytes uit de file gelezen, tenzij er vanaf de opgegeven positie
minder beschikbaar is. Het kanaal wordt zoveel opgeschoven als er werkelijk in bytes is
gelezen en kan als voortijdig het einde van het mapper geheugen wordt gevonden
ongeldig worden gemaakt.

Eventuele disk foutmeldingen kunnen worden opgevraagd via de functie GetError.

Zie ook: MemWriteFile, MemAppendFile

18.4.2.29 MemSavePicture

Declaratie: procedure MemSavePicure(X1, Y1, X2, Y2: Integer;

Page, Channel: Byte);

Gebruik: MemSavePicture(X1,Y1,X2,Y2,Page,Channel);

Variabelen:-

Deze procedure bewaard een plaatje volgens het principe van een COPY-commando van
BASIC. Het plaatje wordt direct vanuit het VRAM bewaard in de memory mapper. De
plaats in de mapper wordt aangegeven door de positie van het opgegeven kanaal
Channel.

Het kanaal nummer mag variÎren van 0 t/m 15. Als het kanaal nummer ongeldig is of het
kanaal stond op een ongeldige positie dan wordt er geen melding teruggegeven. Als er
vanaf het opgegeven kanaal niet genoeg ruimte of exact genoeg ruimte is dan is het
kanaal na afloop ongeldig gemaakt.

Het coˆrdinaat (X1,Y1) is het beginpunt en (X2,Y2) is het eindpunt. Merk op dat er dus in
vier schillende richtingen kan worden gewerkt.

Het weggeschreven plaatje heeft hetzelfde formaat als een file die gemaakt is met het
COPY-commando van BASIC. Het COPY-commando heeft echter de beperking dat een
plaatje niet groter kan zijn als een zichtbaar beeldscherm. Binnen het GIOS is dit echter
geen probleem, er kunnen zelfs plaatjes worden bewaard van 128 Kb.

Zie ook: MemLoadPicture

18.4.2.30 MemWriteFile

Declaratie: procedure MemWriteFile(Channel: Byte; Position: Real;

Size: Real; Name: Str63);

Gebruik: MemWriteFile(Channel,Position,Size,Name);

Variabelen: -

Deze procedure schrijft data naar de opgegeven file beginnend op de opgegeven file
positie. Deze data heeft een grootte van Size bytes en wordt gelezen vanaf de positie
van het opgegeven kanaal.

Deze procedure herschrijft altijd de file waarnaar toe wordt geschreven.

Eventuele disk foutmeldingen kunnen worden opgevraagd via de functie GetError.

Zie ook: MemReadFile, MemAppendFile

18.4.2.31 MoveVRAM

Declaratie: procedure MoveVRAM(SAddress: Integer; SPage: Byte;

DAddress: Integer; DPage: Byte;

Size: Integer);

Gebruik: MoveVRAM(SAddress,SPage,DAddress,DPage,Size);

Variabelen:-

Deze procedure copieert vanaf bron adres SAddress en bron pagina SPage een gebied
met een grootte van Size bytes naar het doel adres SAddress en doel pagina DPage.

Deze functie kan ook gebruikt worden voor het externe VRAM.

18.4.2.32 Paint

Declaratie: procedure Paint(X, Y: Integer; Border: Integer);

Gebruik: PPaint(X,Y,Border);

Variabelen: ActPage, AtrByt, LogOpr

Deze routine kleurt vanaf het co^rdinaat (X,Y) alle pixels in binnen de opgegeven
randkleur met de kleur de is gegeven in de variabele AtrByt. Het resultaat komt op de
pagina in ActPage.

Een logische operatie kan worden opgegeven via de variabele LogOpr. U kunt met deze
procedure ook in het extended VRAM een vlak inkleuren.

Zie ook: DefinePicture, PPaint, PFillShape

18.4.2.33 PFillShape

Declaratie: procedure PFillShape(X, Y: Integer);

Gebruik: PFillShape(X,Y);

Variabelen: ActPage, LogOpr

Deze routine kleurt vanaf het co^rdinaat (X,Y) alle aanliggende pixels met dezelfde kleur
als op het begin co^rdinaat (X,Y) in met de inhoud van de, bij het DefinePicture
genoemde, `achterliggende' scherm. Het resultaat komt op de pagina in ActPage.

Het inkleuren gebeurd d.m.v. GCopy commando's, hierbij kan LogOpr ook gebruikt
worden. Omdat het gebruikte inkleur algoritme alle pixels maar ÈÈn keer bekijkt maakt
het niet uit of een pixel na de GCopy commando's dezelfde kleur krijgt als de kleur
waarnaar gezocht wordt.

Zie ook: DefinePicture, FillShape

18.4.2.34 PPaint

Declaratie: procedure PPaint(X, Y: Integer; Border: Integer);

Gebruik: PPaint(X,Y,Border);

Variabelen: ActPage, LogOpr

Deze routine kleurt vanaf het co^rdinaat (X,Y) alle pixels in binnen de opgegeven
randkleur met de inhoud van de, bij het DefinePicture genoemde, `achterliggende'
scherm. Het resultaat komt op de pagina in ActPage.

Het inkleuren met de `achterliggende' pagina gebeurd met GCopy commando's en daar
bij is LogOpr van invloed.

Zie ook: DefinePicture, Paint, PFillShape

18.4.2.35 PSet

Declaratie: procedure Pset(X: Integer; Y: Byte);

Gebruik: PSet(X,Y);

Variabelen: ActPage, LogOpr, AtrByt

Deze procedure tekent een pixel met kleur AtrByt op het co^rdinaat (X,Y) op de actieve
pagina aangegeven met ActPage. Er kan gebruik gemaakt worden van een logische
operatie LogOpr.

Zie ook: Point

18.4.2.36 PutSprite

Declaratie: procedure PutSprite(X, Y, SpriteNumber: Byte;

PlaneNumber: Byte);

Gebruik: PutSprite(X,Y,SpriteNumber,PlaneNumber);

Variabelen: -

Deze procedure plaatst een sprite figuur op de opgegeven co^rdinaat (X,Y) en met als
patroon nummer SpriteNumber. De sprite wordt geplaatst op het opgegeven PlaneNumber.

Let er op dat de kleuren waarin de sprite vorm wordt weergegeven de kleuren zijn die
ingesteld zijn voor het opgegeven PlaneNumber.

Zie ook: SpritePattern, SpriteAttributeAddress, SpriteColor

18.4.2.37 ReadMem

Declaratie: procedure ReadMem(Channel: Byte; Address: Integer;

Size: Integer);

Gebruik: ReadMem(Channel,Address,Size);

Variabelen: -

Deze procedure leest Size bytes vanaf de positie van het opgegeven kanaal en schrijft
deze naar het opgegeven adres in het werkgeheugen. Het kanaal mag variÎren van 0 t/m
15. Als het kanaal nummer ongeldig is of de positie is ongeldig dan wordt deze
procedure afgebroken.

Na afloop van deze procedure is het betreffende kanaal Size bytes opgeschoven of het
kanaal is ongeldig vanwege het voortijdig bereiken van het einde van het gereserveerde
mapper geheugen.

Er is geen controle of er bij het schrijven naar het werkgeheugen iets overschreven wordt.

Zie ook: WriteMem, GetChannel, SetChannel

18.4.2.38 ReadSector

Declaratie: procedure ReadSector(Drive: Byte; Sector: Integer;
Address: Integer; Sectors:Byte);

Gebruik: ReadSector(Drive,Sector,Address,Sectors);

Variabelen: -

Deze procedure leest een aantal sectoren vanaf de opgegeven drive beginnend op
Sector naar op het gegeven adres. Het aantal te lezen sectoren staat aangegeven in
Sectors. Drive is 0 voor de default drive, 1 voor drive A, 2 voor drive B etc.

Er is geen controle of er door middel van deze procedure ergens iets in het geheugen
wordt overschreven, b.v. het systeem geheugen of de runtime library van Turbo Pascal
zelf. Er is ook geen controle of Sector een nummer is dat binnen het bereik van het
maximaal aantal sectoren valt, dit geldt ook voor de laatste te schrijven sector.

Eventuele disk foutmeldingen kunnen worden opgevraagd via de functie GetError.

Zie ook: WriteSector

18.4.2.39 SavePicture

Declaratie: procedure SavePicure(X1, Y1, X2, Y2: Integer;
Page: Byte; Name: Str63);

Gebruik: SavePicture(X1,Y1,X2,Y2,Page,Name);

Variabelen:-

Deze procedure bewaard een plaatje volgens het principe van een COPY-commando van
BASIC. Het plaatje wordt direct vanuit het VRAM bewaard in de opgegeven file.

Het co^rdinaat (X1,Y1) is het beginpunt en (X2,Y2) is het eindpunt. Merk op dat er dus in
vier schillende richtingen kan worden gewerkt.

Het weggeschreven plaatje heeft hetzelfde formaat als een file die gemaakt is met het
COPY-commando van BASIC. Het COPY-commando heeft echter de beperking dat een
plaatje niet groter kan zijn als een zichtbaar beeldscherm. Binnen het GIOS is dit echter
geen probleem, er kunnen zelfs plaatjes worden bewaard van 128 Kb.

Eventuele disk foutmeldingen kunnen worden opgevraagd via de functie GetError.

Zie ook: LoadPicture

18.4.2.40 Screen

Declaratie: procedure Screen(ScreenNr: Byte);

Gebruik: Screen(ScreenNr);

Variabelen: Border, AtrByt

Deze procedure is gelijk aan het SCREEN-commando van BASIC.

ScreenNr mag variÎren van 0 t/m 12. De schermodes 10 t/m 12 zullen op een MSX-2
hetzelfde resultaat geven als een SCREEN(8).

In tegenstelling tot de procedure ChangeScreen, initialiseerd deze procedure wel het
scherm, eventuele sprite tabellen en wordt de randkleur gewijzigd in de kleur zoals is
opgegeven in Border.

Bij het gebruik van de procedure wordt meteen de viewport aangepast aan de resoluties
van de scherm mode. De linkerbovenhoek wordt altijd teruggezet op co^rdinaat (0,0). De
rechteronderhoek wordt voor de schermodes 6 en 7 op (511,255) gezet en voor alle
andere schermmodes op (255,255).

Het feit of de clipping aan staat wordt door deze procedure niet gewijzigd.

Zie ook: ChangeScreen

18.4.2.41 ScreenOff

Declaratie: procedure ScreenOff;

Gebruik: ScreenOff

Variabelen: -

Deze procedure schakelt het scherm uit waardoor de randkleur over het hele scherm
zichtbaar wordt.

Het scherm uitzetten gebeurd door bit 6 van VDP-register 1 op 0 te zetten.

Omdat de VDP nu meer tijd over heeft voor andere dingen dan de schermweergave zullen
grafische commando's zoals FastBox, FillBox, FastCopy en GCopy merkbaar sneller uitgevoerd worden.

Zie ook: ScreenOn

18.4.2.42 ScreenOn

Declaratie: procedure ScreenOn;

Gebruik: ScreenOn

Variabelen: -

Deze procedure schakelt het scherm dat uit was gezet met de procedure ScreenOff weer aan.

Het scherm uitzetten gebeurd door bit 6 van VDP-register 1 op 1 te zetten.

Zie ook: ScreenOff

18.4.2.43 SetChannel

Declaratie: procedure SetChannel(Channel: Byte; Position: Real);

Gebruik: SetChannel(Channel,Position);

Variabelen: -

Deze procedure zet het opgegeven kanaal Channel op de nieuwe positie Position.

De positie staat aangegeven als een Real, maar deze mag ook als Integer of als Longint worden opgegeven.

Als de Position groter of gelijk is aan SetMem(0) * 16384 dan wordt het betreffende kanaal ongeldig gemaakt. Dit is te zien doordat de functie GetChannel een -1.0 als positie teruggeeft.

Zie ook: GetChannel

18.4.2.44 SetClipping

Declaratie: procedure SetClipping(Flag: Boolean);

Gebruik: SetClipping(Flag);

Variabelen: -

Deze routine zet de clipping aan of uit afhankelijk van Flag. True betekend het aanzetten van de clipping en False betekend het uitzetten. Standaard staat het clippen van commando's uit.

Het clippen van commando's betekend dat een opdracht die voor het hele scherm wordt
gegeven alleen een uitwerking heeft binnen een gebied dat de 'viewport' heet. De grootte
hiervan kan worden ingesteld met de procedure SetViewPort.

Bij de procedure Screen wordt dit viewport altijd opnieuw ingesteld op de grootte van
het scherm, d.w.z. horizontaal van 0 t/m 255 of 0 t/m 511 en verticaal altijd van 0 t/m
255.

Zie ook: SetViewPort

18.4.2.45 SetViewPort

Declaratie: procedure SetViewPort(X1, Y1, X2, Y2: Integer);

Gebruik: SetViewPort(X1,Y1,X2,Y2);

Variabelen: -

Deze procedure stelt de viewport in met behulp van de twee opgegeven coˆrdinaten
(X1,Y1) en (X2,Y2). Hierbij stelt (X1,Y1) de linkerbovenhoek voor en (X2,Y2) de
rechter onderhoek.

Van Y1 en Y2 wordt alleen gekeken naar het lage byte aangezien er geen enkel scherm is
dat meer dan 256 beeldlijnen kan bevatten. Als X1 groter is als X2 dan worden de twee
verwisselt, evenzo voor Y1 en Y2. Bij het ophalen van de beide coˆrdinaten door middel
van GetViewPort krijgt men dan de verwisselde en dus correcte coˆrdinaten terug als
linkerbovenhoek en rechteronderhoek

Zie ook: GetViewPort, SetClipping

18.4.2.46 Sound

Declaratie: procedure Sound(Register, Value: Byte);

Gebruik: Sound(Register, Value);

Variabelen: -

Deze procedure schrijft de opgegeven waarde naar het opgegeven PSG register.

De PSG kent 16 registers, maar hiervan worden alleen register 0 tot en met 13 gebruikt
voor geluidsweergave. De twee overige registers worden gebruikt voor de joystickpoorten.

De procedure Sound is gelijk aan het gelijknamige commando uit BASIC.

Zie ook: ReadPSG

18.4.2.47 SpriteAttributeAddress

Declaratie: procedure SpriteAttributeAddress(Address: Longint);

Gebruik: SpriteAttributeAddress(Address);

Variabelen: -

Deze procedure selecteert ÈÈn van de 128 mogelijke spriteattribuuttabellen.

Als Address een waarde is lager dan 128 dan wordt ervan uitgegaan dat het gaat om
een nummer van de betreffende spritepatroontabel. Als Address een waarde is die
groter is dan 127 dan wordt ervan uitgegaan dat het gaat om een adres.

Het adres wordt berekend met de formule: (Address and $1FC00)+512. Dit adres
begint op 512 en gaat in stappen van 1024 bytes omhoog tot een maximum van 130560
decimaal of $1FE00 hexadecimaal, respectievelijk spriteattribuuttabel 0 t/m 127.

De spriteattribuuttabel is altijd 128 bytes lang en alleen in de scherm modes 4 t/m 8 en 10
t/m 12 is deze spritekleurtabel aanwezig. Deze tabel bevindt zich altijd 512 bytes lager
dan de spriteattribuuttabel.

Zie ook: SpritePatternAddress, PutSprite

18.4.2.48 SpriteColor

Declaratie: procedure SpriteColor(PlaneNumber: Byte;

Address: Integer);

Gebruik: SpriteColor(PlaneNumber,Address);

Variabelen: -

Deze procedure vult de kleurdata behorende bij het opgegeven plane nummer met de data
die staat in het werkgeheugen op Address.

PlaneNumber varieert van 0 t/m 31 voor het maximaal aantal aanwezige zichtbare
sprites. De kleurdata wordt geschreven naar de spritekleurtabel die zich altijd 512 bytes
lager bevindt dan de spriteattribuuttabel.

Zie ook: FillSprite, SpriteAttributeAddress

18.4.2.49 SpritePattern

Declaratie: procedure SpritePattern(SpriteNumber: Byte;

Address: Integer);

Gebruik: SpritePattern(SpriteNumber,Address);

Variabelen: -

Deze procedure vult de vorm van de opgegeven sprite in de spritepatroontabel met de
patroon data die staat op het opgegeven adres Address in het werkgeheugen.

Afhankelijk van de opgegeven grootte die bij de procedure SpriteSize wordt
ingevuld gaat het om 8 bytes voor een 8x8 sprite of 32 bytes voor een 16x16 sprite.
SpriteNumber varieert voor 8x8 sprites van 0 t/m 255 en voor 16x16 sprites van

0 t/m 63.

Zie ook: SpritePatternAddress

18.4.2.50 SpritePatternAddress

Declaratie: procedure SpritePatternAddress(Address: Longint);

Gebruik: SpritePatternAdress(Address);

Variabelen: -

Deze procedure selecteert ÈÈn van de 64 mogelijke spritepatroontabellen.

Als Address een waarde is lager als 64 dan wordt ervan uitgegaan dat het gaat om het
nummer van de betreffende spritepatroontabel. Als Address een waarde is die groter is
dan 63 dan wordt ervan uitgegaan dat het gaat om een adres. Dit adres begint op 0 en
gaat in stappen van 2048 bytes omhoog tot een maximum van 129024 decimaal of
$1F800 hexadecimaal, respectievelijk spritepatroontabel 0 t/m 63. De spritepatroontabel is altijd 2048 bytes lang.

Zie ook: SpriteAttributeAddress, PutSprite

18.4.2.51 SpriteSize

Declaratie: procedure SpriteSize(Size: Byte);

Gebruik: SpriteSize(Size);

Variabelen: -

Deze procedure stelt de grootte van de sprites in de VDP in. Size kan de volgende
waarden hebben:

Size

Weergave

0

sprites 8 * 8 normaal

1

sprites 8 * 8 vergroot

2

sprites 16 * 16 normaal

3

sprites 16 * 16 vergroot

De vergroting is 2 maal, zowel in de hoogte als in de breedte.

18.4.2.52 SpritesOff

Declaratie: procedure SpritesOff;

Gebruik: SpritesOff;

Variabelen: -

Deze procedure zet de sprites uit. Het uitzetten van de sprites heeft geen enkel effect op de sprites zelf en/of de bijbehorende data. Ook hier is sprake van een versnelling van een aantal grafische commando's, net zoals bij ScreenOff, maar deze is heel klein in vergelijking tot ScreenOff.

De opdracht:

```
WriteVDP(8,ReadVDP(8) or 2);
```

doet precies hetzelfde.

Zie ook: SpritesOn

18.4.2.53 SpritesOn

Declaratie: procedure SpritesOn;

Gebruik: SpritesOn;

Variabelen: -

Deze procedure zet de sprites aan.

De opdracht:

```
WriteVDP(8,ReadVDP(8) and $FD);
```

doet precies hetzelfde.

Zie ook: SpritesOff

18.4.2.54 Time

Declaratie: procedure Time(var Hour, Minute, Second: Byte);

Gebruik: Time(Hour,Minute,Second);

Variabelen: -

Deze procedure leest de huidige tijd uit de klokchip.

Bij MSX-DOS 2 zal de tijd in een 12 of 24 uurs notatie komen te staan, voor MSX-DOS
1 zal de tijd altijd in een 24 uurs notatie komen te staan.

Zie ook: SetTime

18.4.2.55 VPoke

Declaratie: procedure VPoke(Address: Integer; Data: Byte);

Gebruik: VPoke(Address,Data);

Variabelen: ActPage

Deze procedure schrijft een byte op het opgegeven VRAM Address op de pagina die
staat in ActPage.

Deze procedure kan ook data schrijven naar het externe VRAM.

Zie ook: VPeek

18.4.2.56 WaitVDP

Declaratie: procedure WaitVDP;

Gebruik: WaitVDP;

Variabelen: -

Deze procedure wacht totdat een VDP-commando klaar is. Een commando dat gebruik
maakt van een VDP-commando wacht altijd eerst tot dat het vorige commando klaar is.
De volgende statements gebruiken deze VDP-commando's:

FillShape

PFillShape

Paint

PPaint

Line

Search

Point

PSet

FastBox

FillBox

FastCopy

GCopy

Circle

Ellipse

Expand

MemExpand

LoadPicture

MemLoadPicture

Expand

MemExpand

LoadPicture

MemLoadPicture

SavePicture

MemSavePicture

Screen, in ieder geval
voor screen 10 t/m 12

De volgende statements hebben wel betrekking op het VRAM maar gebruiken geen
VDP-commando's en dus geen WaitVDP:

Vpeek

Vpoke

Bload

Bsave

FillSprite

PutSprite

SpriteColor

SpritePattern

LoadVRAM

MoveVRAM

Als een commando uit de bovenste tabel nog niet klaar is en men gebruikt ÈÈn van de
commando's uit de onderste tabel, dat in hetzelfde gebied werkt als het eerste commando,
dan is de kans groot dat deze twee commando's elkaar dwars zitten. Het kan ook zijn dat
de Z80 op 3.57 Mhz te traag is om het vorige statement dat een VDP-commando gebruikt
heeft dwars te zitten, maar dat het op een hogere snelheid (b.v. 7 Mhz of een TURBO-R)
het VDP-commando wel dwars zit. Dit zijn vrij lastige fouten en om dit soort conflicten
te voorkomen moet men een WaitVDP gebruiken.

18.4.2.57 WriteMem

Declaratie: procedure WriteMem(Channel: Byte;

Address: Integer; Size: Integer);

Gebruik: WriteMem(Channel,Address,Size);

Variabelen: -

Deze procedure schrijft Size bytes vanaf het opgegeven adres in het werkgeheugen naar de positie van het opgegeven kanaal. Het kanaal mag variÎren van 0 t/m 15. Als het kanaal nummer ongeldig is of de positie is ongeldig dan wordt deze procedure afgebroken.

Na afloop van deze procedure is het betreffende kanaal Size bytes opgeschoven of het kanaal is ongeldig vanwege het voortijdig bereiken van het einde van het gereserveerde mapper geheugen.

Zie ook: ReadMem, GetChannel, SetChannel

18.4.2.58 WriteSector

Declaratie: procedure WriteSector(Drive: Byte; Sector: Integer;

Address: Integer;Sectors:Byte);

Gebruik: WriteSector(Drive,Sector,Address,Sectors);

Variabelen: -

Deze procedure schrijft een aantal sectoren vanaf Address naar de opgegeven drive beginnend op Sector. Het aantal te schrijven sectoren staat aangegeven in Sectors.
Drive is 0 voor de default drive, 1 = drive A, 2 voor drive B etc.

Er is ook geen controle of Sector een nummer is dat binnen het bereik van het maximaal aantal sectoren valt, dit geldt ook voor de laatste te schrijven sector.

Eventuele disk foutmeldingen kunnen worden opgevraagd via de functie GetError.

Zie ook: ReadSector

18.4.2.59 WriteVDP

Declaratie: procedure WriteVDP(Register, Data: Byte);

Gebruik: WriteVDP(Register, Data);

Variabelen: -

Deze procedure schrijft de opgegeven data naar een register van de VDP.

Er wordt altijd data naar het opgegeven register geschreven ook al is het register voor de V9938/V9958 een niet bestaand register. De data wordt alleen in het systeem geheugen weggeschreven als het register nummer een geldig nummer is. Zie voor de geldige nummers de functie ReadVDP. U kunt ook de registers 32 t/m 46 beschrijven, dit zijn de registers die gebruikt worden voor speciale commando's zoals b.v. FastCopy of Line etc. Let er op dat als het register groter of gelijk is aan 128 dat men geen register beschrijft maar dat men een VRAM adres klaar zet.

Zie ook: ReadVDP

APPENDIX A

Overzicht van de standaard procedures en functies

In deze appendix staan alle standaard procedures en functies opgesomd die in Turbo Pascal beschikbaar zijn. Hierin kunt u opzoeken hoe de schrijfwijze is, waarvoor ze worden gebruikt en de eventuele parameters. De volgende symbolen zijn gebruikt om aan te geven van welk type een parameter moet zijn:

type een willekeurig type

string een willekeurig String type (inclusief Char)

file een willekeurig File type

scalair een willekeurig scalair type (Byte, Integer)

pointer een willekeurig wijzer type

Als er geen type bij een parameter staat vermeld betekend dit dat de procedure of functie
elk type accepteert.

Invoer en uitvoer procedures en functies

De volgende procedures gebruiken een schrijfwijze die niet standaard is in hun
parameterlijst:

```
procedure

   Read   (var F: File of type; var V: type);

   Read   (var F: Text; var I: Integer);

   Read   (var F: Text; var L: Longint);

   Read   (var F: Text; var R: Real);

   Read   (var F: Text; var C: Char);

   Read   (var F: Text; var S: string);

   ReadLn (var F: Text);

   Write  (var F: File of type; var V: type);

   Write  (var F: Text; I: Integer);

   Write  (var F: Text; L: Longint);

   Write  (var F: Text; R: Real);

   Write  (var F: Text; B: Boolean);

   Write  (var F: Text; C: Char);

   Write  (var F: Text; S: string);

   WriteLn(var F: Text);
```

Rekenkundige functies

```
function

   Abs (I: Integer): Integer;

   Abs (L: Longint): Longint;

   Abs (R: Real): Real;
```

```
   ArcTan (R: Real): Real;
   Cos (R: Real): Real;
   Exp (R: Real): Real;
   Frac (R: Real): Real;
   Int (R: Real): Real;
   Ln (R: Real): Real;
   Sin (R: Real): Real;
   Sqr (I: Integer): Integer;
   Sqr (L: Longint): Longint;
   Sqr (R: Real): Real;
   Sqrt (R: Real): Real;
```

Scalaire functies

```
function
   Odd  (I: Integer): Boolean;
   Odd  (L: Longint): Boolean;
   Pred (X: scalair): scalair;
   Succ (X: scalair): scalair;
```

Converteer functies

```
function
   Chr   (I: Integer): Char;
   Chr   (L: Longint): Char;
   Ord   (X: scalair): Integer;
   Round (R: Real): Integer;
   Trunc (R: Real): Integer;
```

Procedures en functies voor string bewerkingen

De Str procedure gebruikt een niet standaard schrijfwijze voor zijn numerieke

parameter.

```
procedure
   Delete (var S: string; Pos, Len: Integer);
   Insert (S: string; var D: string; Pos: Integer);
   Str (I: Integer; var S: string);
   Str (L: Longint; var S: string);
   Str (R: Real; var S: string);
   Val (S: string; var R: Real; var P: Integer);
   Val (S: string; var L: Longint; var P: Integer);
   Val (S: string; var I, P: Integer);

function
   Concat (S1,S2,...,Sn: string): string;
   Copy   (S: string; Pos, Len: Integer): string;
   Length (S: string): Integer;
   Pos    (Vorm, Bron: string): Integer;
```

Procedures en functies voor filebewerking

```
procedure
   Append  (var F: file; Naam: string);
   Assign  (var F: file; Naam: string);
   BlockRead (var F: file; var Doel: type; Num: Integer);
   BlockWrite(var F: file; var Doel: type; Num: Integer);
   Chain   (var F: file);
   Close   (var F: file);
   Erase   (var F: file);
   Execute (var F: file);
   Rename  (var F: file; Naam: string);
```

```
   Reset   (var F: file [; RecordSize: Integer]);
   Rewrite (var F: file [; RecordSize: Integer]);
   Seek    (var F: file of type; Pos: Longint);

function
   Eof     (var F: file): Boolean;
   Eoln    (var F: Text): Boolean;
   FilePos (var F: file): Longint;
   FileSize(var F: file of type): Longint;
   FileSize(var F: file): Longint;
   SeekEof (var F: file): Boolean;
   SeekEoln(var F: Text): Boolean;
```

Procedures en functies voor heap beheer

```
procedure
   Dispose (var P: pointer);
   FreeMem (var P: pointer; I: Integer);
   GetMem  (var P: pointer; I: Integer);
   Mark    (var P: pointer);
   New     (var P: pointer);
   Release (var P: pointer);

function
   MaxAvail: Integer;
   MemAvail: Integer;
   Ord (P: pointer): Integer;
   Ptr (I: Integer): pointer;
```

Procedures en functies voor scherm aansturing;

```
procedure
   CrtExit;
   CrtInit;
   ClrEol;
   ClrScr;
   DelLine;
   GotoXY (X, Y: Integer);
   InsLine;
   LowVideo;
   NormVideo;
```

Diverse procedures en functies

```
procedure
   Bdos (Func, Param: Integer);
   Bios (Func, Param: Integer);
   Delay (mSec: Integer);
   FillChar (var Doel, Lengte: Integer; Data: Char);
   FillChar (var Doel, Lengte: Integer; Data: Byte);
   Halt;
   Move (var Bron, Doel: type; Lengte: Integer);
   Randomize;

function
   Addr (var Variabele): Integer;
   Addr (<functie naam>): Integer;
   Addr (<procedure naam>): Integer;
   Bdos (Func, Param: Integer): Byte;
   BdosHL (Func, Param: Integer): Integer;
   Bios (Func, Param: Integer): Byte;
```

```
BiosHL (Func, Param: Integer): Integer;

Hi (I: Integer): Integer;

Hi (L: Longint): Longint;

IOResult: Integer;

KeyPressed: Boolean;

Lo (I: Integer): Integer;

ParamCount: Integer;

ParamStr (N: Integer): string;

Random (Bereik: Integer): Integer;

Random: Real;

SizeOf (var Variabele): Integer;

SizeOf ( type ): Integer;

Swap (I: Integer): Integer;

UpCase (Ch: Char): Char;
```

APPENDIX B

Overzicht van bewerkingen

De volgende tabel toont alle bewerkingen in Turbo Pascal. De operatoren zijn geordend
in aflopende voorrangsvolgorde. Waar het soort operand is omschreven als 'als operand'
is het resultaat als volgt:

Operand

Resultaat

Integer, Integer

Integer

Integer, Longint

Longint

Longint, Longint

Longint

Real, Real

Real

Real, Integer

Real

Real, Longint

Real

Operator

Bewerking

Operand type

Resulterend type

+ monadisch

teken bevestiging

Integer, Longint, Real

als operand

- monadisch

teken omkeren

Integer, Longint, Real

als operand

not

inverteren

Boolean,Integer,Longint

als operand

*

vermenigvuldiging

Integer, Longint, Real

Integer, Longint, Real

doorsnede van sets

een set type

als operand

/

deling

Integer, Longint, Real

Real

div

gehele deling

Integer, Longint

Integer, Longint

mod

modulo

Integer, Longint

Integer, Longint

and

rekenkundige and

Integer, Longint

Integer, Longint

logische and

Boolean

Boolean

shl

schuif links

Integer, Longint

Integer, Longint

shr

schuif rechts

Integer, Longint

Integer, Longint

+

optellen

Integer, Longint, Real

Integer, Longint, Real

samenvoeging

String

String

vereniging van sets

een set type

als operand

-

aftrekken

Integer, Longint, Real

Integer, Longint, Real

verschil van sets

een set type

als operand

or

rekenkundige or

Integer, Longint

Integer, Longint

logische or

Boolean

Boolean

xor

rekenkundige xor

Integer, Longint

Integer, Longint

logische xor

Boolean

Boolean

=

gelijkheid

een scalair type

Boolean

gelijkheid

String

Boolean

gelijkheid

een set type

Boolean

gelijkheid

een pointer type

Boolean

<>

ongelijkheid

een scalair type

Boolean

ongelijkheid

String

Boolean

ongelijkheid

een set type

Boolean

ongelijkheid

een pointer type

Boolean

>=

groter of gelijk

een scalair type

Boolean

groter of gelijk

String

Boolean

groter of gelijk

een set type

Boolean

<=

kleiner of gelijk

een scalair type

Boolean

kleiner of gelijk

String

Boolean

kleiner of gelijk

een set type

Boolean

>

groter als

een scalair type

Boolean

groter als

String

Boolean

<

kleiner als

een scalair type

Boolean

kleiner als

String

Boolean

in

vervat in set

zie hieronder

Boolean

De eerste operand bij de in operator mag van elk scalair type zijn, de tweede operand
moet een set zijn van hetzelfde type.

APPENDIX C

Overzicht compiler aanwijzingen

Een aantal opties van de Turbo Pascal compiler worden ingesteld met de compiler
aanwijzingen. De compiler aanwijzingen zijn in de code opgenomen als commentaar met
een speciale vorm. Dit betekend dat overal waar commentaar mag staan ook een compiler
aanwijzing mag staan.

Een compiler aanwijzing bestaat uit een openingshaak { gevolgd door een dollarteken
wat weer gevolgd wordt door een letter of een lijst met compiler aanwijzing letters
gescheiden door een punt. Tot slot volgt de afsluitende haak }.

Voorbeelden:

```
{$I-}

{$I INCLUDE.FIL}

{$B-,R+,V-}

(*$U+*)
```

Zoals hierboven te zien is, mag ook als alternatief het commentaar beginnen met (* en
eindigen met *). Merk op dat er geen spaties mogen staan voor of achter het dollarteken.

Een plus teken geeft aan dat de compiler aanwijzing is aangezet (actief) en een min teken
geeft aan dat de aanwijzing is uitgeschakeld (passief).

Belangrijke aantekening

Alle compiler aanwijzingen hebben een vooraf ingestelde waarde. Deze instelling is zo
gekozen dat er een optimale verwerkingssnelheid ontstaat en de code zo klein mogelijk
blijft. Dit betekend onder andere dat index controle is uitgezet en code generatie voor
recursieve procedures en functies ook uit staat. Controleer dus of de opties voor uw
programma goed staan ingesteld.

B - I/O mode keuze

Standaard: B+

De B compiler aanwijzing controleert de keuze van de I/O mode. Als deze actief is
{$B+} wordt het CON: apparaat gekoppeld aan de standaard files Input en Output. Als
de compiler aanwijzing passief is {$B-}, wordt het TRM: apparaat gebruikt.

Deze compiler aanwijzing is voor het hele programma en kan niet halverwege opnieuw
worden gedefinieerd. Zie hoofdstuk 13.7, 13.8 en 13.9 voor meer details.

C - CTRL-C en CTRL-S

Standaard: C+

De C compiler aanwijzing controleert de interpretatie tijdens I/O met het
bedieningspaneel. Als de optie actief {$C+} is, kan tijdens een Read of een ReadLn met
de toetscombinatie CTRL-C het programma worden afgebroken en met CTRL-S kan
beeldschermuitvoer worden gepauzeerd. Wanneer de optie passief {$C-} is, wordt er
niet gecontroleerd op deze combinaties.

Als de optie actief is wordt de schermuitvoer iets vertraagd. Als snelle schermuitvoer een
vereiste is dient u deze aanwijzing dan ook passief te maken.

Deze compiler aanwijzing is voor het hele programma en kan niet halverwege opnieuw
worden gedefinieerd.

I - I/O fout afhandeling

Standaard: I+

De I compiler aanwijzing controleert de I/O fout afhandeling. Als de optie actief is
{$I+} worden alle I/O handelingen gecontroleerd door het systeem. Als de optie passief
is {$I-} moet de controle gebeuren door de programmeur met behulp van de standaard
functie IOResult. Zie hoofdstuk 13.4 "Het gebruik van files" voor meer details.

I - Tussenvoeg files (include files)

De I compiler aanwijzing wordt gevolgd door een file naam. De compiler zal op dit punt
een nieuwe file inlezen en deze eerst vertalen. Deze aanwijzig is in Turbo Pascal versie
3.3 uitgebreid. Het is toegestaan om een complete drive/pad/naam op te geven en men
mag tot maximaal 4 niveau's diep files includen.

R - Index bereik controle

Standaard: R-

De R compiler aanwijzing controleert tijdens het uitvoeren van een programma of de
opgegeven indices van arrays niet buiten het bereik vallen. Als de compiler aanwijzing
actief is, {$R+}, worden alle indexen gecontroleerd en van alle toekenningen aan
scalairen wordt gecontroleerd of de toekenning wel binnen het bereik van de scalaire
variabele valt. Als de compiler aanwijzing niet actief is, {$R-}, worden er geen extra
controles uitgevoerd. Het is verstandig om bij de ontwikkeling van een programma deze
compiler aanwijzing actief te maken.

V - Var-parameter type controle

Standaard: V+

De V compiler aanwijzing controleert de type controle van Strings die worden opgeven
als var parameters. Als de compiler aanwijzing actief is, {$V+}, wordt er een zeer strikte
controle uitgevoerd, d.w.z. dat de lengte van de actuele parameter en de formele
parameter gelijk moeten zijn. Als de compiler aanwijzing niet actief is, {$V-}, is het
toegestaan dat de lengte van de actuele parameter niet gelijk is aan de lengte van de
formele parameter.

U - User interrupt

Standaard: U-

De U compiler aanwijzing controleert de interrupts van de gebruiker. Als de compiler aanwijzing actief is, {$U+}, kan de gebruiker op ieder tijdstip het programma onderbreken d.m.v. een CTRL-C toetscombinatie. Als de compiler aanwijzing niet actief is, {$U-}, heeft een CTRL-C geen effect. Het aanzetten van deze compiler aanwijzing geeft een behoorlijk performance verlies.

A - Absolute code generatie

Standaard: A+

De A compiler aanwijzing controleert de generatie van absolute code of de generatie van recursieve code. Als compiler aanwijzing actief is, {$A+}, wordt er absolute code gegenereerd. Als de compiler aanwijzing niet actief is, {$A-}, wordt er code gegenereerd waardoor recursieve aanroepen van procedures en functies zijn toegestaan. Deze recursieve code neemt meer geheugen in beslag en is langzamer in de uitvoering.

W - Maximale nesting diepte van with statements

Standaard: W2

De W compiler aanwijzing controleert de maximale nesting diepte van with statements, dit is het aantal records dat tegelijk 'geopend' kan zijn binnen een with statement. De W moet direct gevolgd worden door een cijfer tussen de 1 en de 9.

X - Array optimalisatie

Standaard: X+

De X compiler aanwijzing controleert de optimalisatie. Als de compiler aanwijzing actief is, {$X+}, wordt code voor de index berekening van arrays geoptimaliseerd voor een maximale snelheid. Als de compiler aanwijzing niet actief is, {$X-}, wordt de gegenereerde voor de arrays zo klein mogelijk gehouden.

APPENDIX D

RUN-TIME Foutmeldingen

Hieronder volgt een lijst met foutmeldingen die kunnen optreden tijdens het uitvoeren van een programma.

Als de foutmeldingen niet binnen het programma worden afgevangen zal er worden
teruggekeerd naar DOS. Hierbij wordt scherm 0 (tekstscherm) actief gemaakt. De
foutmelding ziet er als volgt uit:

```
Run-time error NN, PC=addr

Program aborted
```

Hierbij is NN het nummer van de foutmelding en addr is het adres in de programma code
waar de foutmelding optrad. Let er op dat de nummers hexadecimaal zijn weergegeven.

01

Floating point overflow.

02

Division by zero attempted.

03

Sqrt argument error.

Het argument naar de SQRT functie bevat een negatieve waarde.

04

Ln argument error.

Het argument naar de LN functie bevat een nul of een negatieve waarde.

10

String length error.

1) Een string samenvoeging leidde tot een string die langer is als 255 karakters.

2) alleen strings met een lengte van 1 kunnen worden omgezet naar karakters.

11

Invalid string index.

De index expressie blijft niet binnen 1..255 bij de procedure aanroep van Copy,
Delete of Insert.

90

Index out of range.

De index expressie bij het benaderen van een array viel buiten het bereik.

91

Scalar or subrange out of range.

De waarde die toegekend werd aan een scalair of een deelgebied type viel buiten het bereik.

92

Out of integer range.

De real waarde die doorgegeven werd aan Trunc of Round viel niet binnen het
integer bereik -32768..32767.

AA

MEMMAN not present.

AB

Wrong MEMMAN version.

AC

GIOS TSR not present.

AD

GIOS not present.

AE

Wrong GIOS version.

F0

Overlay file not found.

FF

Heap/stack collision.

APPENDIX E

I/O Foutmeldingen

Een fout in een input of output operatie tijdens het uitvoeren van een programma
resulteert in een I/O error. Als de I/O controle (I compiler directieve) aan staat dan stopt
het programma en wordt de volgende melding weergegeven:

```
I/O error error NN, PC=addr

Program aborted
```

Hierbij is NN het nummer van de foutmelding en addr is het adres in de programma code
waar de foutmelding optrad. Let er op dat de nummers hexadecimaal zijn weergegeven.

01

File does not exist

De gebruikte file naam in de standaard procedures Reset, Erase, Rename, Execute of Chain bestaat niet.

02

File not open for input

1) Er wordt geprobeerd te lezen met een Read of ReadLn van een file zonder

een voorafgaande Reset of ReWrite.

2) Er wordt geprobeerd te lezen van een text file die vooraf geopend was met

een ReWrite en dus leeg is.

3) Er wordt geprobeerd te lezen van een de printer (LST:).

03

File not open for output

1) Er wordt geprobeerd te schrijven met een Write of WriteLn naar een file

zonder een voorafgaande Reset of ReWrite.

2) Er wordt geprobeerd te schrijven naar een text file die vooraf geopend was

met een Reset.

3) Er wordt geprobeerd te schrijven naar het toetsenbord (KBD:).

04

File not open

Er wordt geprobeerd te lezen of te schrijven met een BlockRead of BlockWrite
zonder vooraf de file te openen met een Reset of een ReWrite.

10

Error in numeric format

De gelezen string van een text file naar een numerieke variabele is niet conform
het formaat van deze numerieke variabele.

20

Operation not allowed on a logical device

Er wordt geprobeerd een Erase, Rename, Execute of een Chain procedure uit te voeren met een file die is toegekend aan een logisch device.

21

Not allowed in direct mode

Niet meer van toepassing in Turbo Pascal versie 3.3.

22

Assign to std files not allowed

De standaard file variabele 'input' of 'output' kan niet worden gebruikt in een
Assign procedure.

90

Record length mismatch

De record lengte die aanwezig is in de file komt niet overeen met de record
lengte in de opgegeven getypeerde file variabele.

91

Seek beyond end-of-file

99

Unexpected end-of-file

1) Het einde van een text file is gevonden zonder dat een voorafgaand EOF

character (Ctrl-Z) is gevonden.

2) Er wordt geprobeerd te lezen voorbij het einde van een getypeerde file.

F0

Disk write error

De disk is vol.

F1

Directory is full

Er is geen ruimte meer in de directory voor een nieuwe file.

F2

File size overflow

Er wordt geprobeerd een record te schrijven voorbij de grens van 65535 records in een getypeerde file.

F3

Too many open files

Er zijn teveel files tegelijkertijd geopend.

FF

File disappeared

Er wordt geprobeerd een file te sluiten die niet langer aanwezig is.

B.v. de disk is verwisseld.

APPENDIX F

Vertaling van Foutmeldingen

Hieronder volgt een lijst met foutmeldingen die kunnen optreden tijdens het compileren.

01

';' expected

02

':' expected

03

',' expected

04

'(' expected

05

')' expected

06

'=' expected

07

':=' expected

08

'[' expected

09

']' expected

10

'.' expected

11

'..' expected

12

BEGIN expected

13

DO expected

14

END expected

15

OF expected

16

PROCEDURE or FUNCTION expected

17

THEN expected

18

TO or DOWNTO expected

19

Byte variable expected

20

BOOLEAN expression expected

21

File variable expected

22

Integer constant expected

23

Integer expression expected

24

Integer variable expected

25

Integer or real constant expected

26

Integer or real expression expected

27

Integer or real variable expected

28

Pointer variable expected

29

Record variable expected

30

Simple type expected

Alle scalaire typen, met uitzondering van real.

31

Simple expression expected

32

String constant expected

33

String expression expected

34

String variable expected

35

Textfile expected

36

Type identifier expected

37

Untyped file expected

40

Undefined label

Er wordt een nog niet gedefinieerd label gebruikt.

41

Unknown identifier or syntax error

Onbekend label, constante, type, variabele of veld naam. Ook kan er een schrijffout in een opdracht zitten.

42

Undefined pointer type in preceding type definitions

Er is een pointer gedefinieerd naar een type, terwijl dit type niet is gedefinieerd.

43

Duplicate identifier or label

De gebruikte naam is al eens gedefinieerd binnen dit blok.

44

Type mismatch

1) het type van de gebruikte variabele en de expressie komen niet overeen,

2) de parameter naar een procedure of functie is van het verkeerde type,

3) het type van de expressie komt niet overeen met het indextype van een array,

4) binnen een expressie komen de typen van de operanden niet overeen.

45

Constant out of range

Een constante valt buiten het toegestane bereik.

46

Constant and CASE selector type does not match

47

Operand type(s) does not match operator

Voorbeeld: 'A' div '2'.

48

Invalid result type

Geldige typen zijn alle scalaire typen, string typen en pointer typen.

49

Invalid string length

De lengte van een string moet binnen 1..255 vallen.

50

String constant length does not match type

51

Invalid subrange base type

Geldige basis typen zijn alle scalaire typen met uitzondering van real.

52

Lower bound > upper bound

De ondergrens van een deelbereik is groter dan de bovengrens.

53

Reserved word

Gereserveerde woorden mogen niet worden gebruikt in namen.

54

Illegal assignment

Er wordt een toekenning gemaakt waarbij de typen niet overeenkomen.

55

String constant exceeds line

String constanten moeten op ÈÈn regel worden geschreven.

56

Error in integer constant

Een integer constante voldoet niet aan de schrijfwijze zoals deze is omschreven in hoofdstuk 2.1 "Integer (hele getallen)" of de constante blijft niet binnen het bereik -32768..32767. Gehele reÎle getallen moeten worden geschreven met een punt gevolgd door een nul. Bijvoorbeeld 123456789.0.

57

Error in real constant

De schrijfwijze van de real staat beschreven in hoofdstuk 2.3 "Real (reÎle getallen)".

58

Illegal character in identifier

60

Constants are not allowed here

61

Files and pointers are not allowed here

62

Structured variables are not allowed here

63

Textfiles are not allowed here

64

Textfiles and untyped files are not allowed here

65

Untyped files are not allowed here

66

I/O not allowed here

67

Files must be VAR parameters

68

File components may not be files

De constructie: file of file is niet toegestaan.

69

Invalid ordering of fields

70

Set base type out of range

Het basis type van een set moet een scalair type zijn met niet meer dan 256
mogelijke waarden.

71

Invalid GOTO

72

Label not within current block

Een GOTO kan nooit buiten het huidige blok springen.

73

Undefined FORWARD procedure(s)

Een procedure of functie is met FORWARD gedeclareerd maar het blok met de eigenlijke procedure of functie is niet aanwezig.

74

Inline error

75

Illegal use of ABSOLUTE

1) er mag maar ÈÈn naam staan voor de dubbele punt in een absolute variabele

declaratie.

2) de ABSOLUTE declaratie mag niet binnen een record worden gebruikt.

76

Overlays can not be forwarded

De FORWARD declaratie kan niet worden gebruikt in combinatie met overlays.

77

Overlays not allowed in direct mode

Deze melding is vervallen vanaf Turbo Pascal versie 3.3

90

File not found

De aangegeven include file kon niet worden gevonden.

91

Unexpected end of source

Het programma eindigt niet op een logische manier. Waarschijnlijk staat er
vaker een BEGIN dan een END in uw source.

92

Unable to create overlay file

93

Invalid compiler directive

96

Too many nested files

Het maximale diepte van vier include files is overschreden.

97

Too many nested WITHs

Gebruik de W compiler aanwijzing om het maximaal geneste WITHs te verhogen.

98

Memory overflow

Er wordt geprobeerd meer geheugen te reserveren voor variabelen als er beschikbaar is.

99

Compiler overflow

Er is niet genoeg geheugen aanwezig om de compilatie te kunnen uitvoeren.

C8

Run-time error found

FA

Disk or directory full

+

De com-file kon niet worden gecreÎerd vanwege een te volle directory of een

APPENDIX G

Turbo Pascal SYNTAX

Turbo Pascal syntax:

```
actual-parameter ::= expression | variable

adding-operator ::= + | - | or | xor

array-constant ::= ( structured-constant {, structured-constant } )

array-type ::= array [ index-type {, index-type } ] of component-type

array-variable ::= variable

assignment-statement ::= variable := expression |
                         function-identifier ::= expression


base-type ::= simple-type

block ::= declaration-part statement-part

case-element ::= case-list : statement

case-label ::= constant

case-label-list ::= case-label {, case-label }

case-list ::= case-list-element {, case-list-element }

case-list-element ::= constant | constant .. constant

case-statement ::= case expression of case-element {; case-element } end
|
                   case expression of case-element {; case-element }
```

```
                    otherwise statement {; statement} end


complemented-factor ::= signed-factor | not signed-factor

component-type ::= type

component-variable ::= indexed-variable | field-designator

compound-statement ::= begin statement {; statement } end

conditional-statement ::= if-statement | case-statement


constant ::= unsigned-number | sign unsigned-number | constant-identifier
|
             sign constant-identifier | string

constant-definition-part ::= const constant-definition

                             {; constant-definition };

constant-definition ::= untyped-constant-definition |

                        typed-constant-definition

constant-identifier ::= identifier

control-character ::= # unsigned-integer | ^ character

control-variable ::= variable-identifier

declaration-part ::= { declaration-section }

declaration-section ::= label-declaration-part | constant-definition-part
|
                        type-definition-part | variable-declaration-part
|
                        procedure-and-function-declaration-part

digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

digit-sequence ::= digit { digit }

empty ::=

empty-statement ::= empty

entire-variable ::= variable-identifier | typed-constant-identifier

expression ::= simple-expression { relational-operator simple-expression
}
```

```
factor ::= variable | unsigned-constant | ( expression ) |
           function-designator | set

field-designator ::= record-variable . field-identifier

field-identifier ::= identifier

field-list ::= fixed-part | fixed-part ; variant-part | variant-part

file-identifier ::= identifier

file-identifier-list ::= empty | ( file-identifier {, file-identifier }

file-type ::= file of type

final-value ::= expression

fixed-part ::= record-section {; record-section }

for-list ::= initial-value to final-value | initial-value downto final-
value

for-statement ::= for control-variable := for-list do statement

formal-parameter-section ::= parameter-group | var parameter-group

function-declaration ::= function-heading block ;

function-designator ::= function-identifier | function-identifier
                        ( actual-parameter {, actual-parameter } )

function-heading ::= function identifier : result-type ; |
                     function identifier ( formal-parameter-section
                     {, formal-parameter-section } ) : result-type ;

function-identifier ::= identifier ;

goto-statement ::= goto label

hexdigit ::= digit | A | B | C | D | E | F

hexdigit-sequence ::= hexdigit { hexdigit }

identifier ::= letter { letter-or-digit }

identifier-list ::= identifier {, identifier }

if-statement ::= if expression then statement { else statement }

index-type ::= simple-type

indexed-variable ::= array-variable [ expression {, expression} ]

initial-value ::= expression
```

```
inline-list-element ::= unsigned-integer | constant-identifier |
                        variable-identifier | location-counter-reference

inline-statement ::= inline inline-list-element {, inline-list-element }

label ::= letter-or-digit { letter-or-digit }

label-declaration-part ::= label label {, label } ;

letter ::= A | B | C | D | E | F | G | H | I | J | K | L | M |
           N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
           a | b | c | d | e | f | g | h | i | j | k | l | m |
           n | o | p | q | r | s | t | u | v | w | x | y | z | _

letter-or-digit ::= letter | digit

location-counter-reference ::= * | * sign constant

multiplying-operator ::= * | / | div | mod | and | shl | shr

parameter-group ::= identifier-list : type-identifier

pointer-type ::= ^ type-identifier

pointer-variable ::= variable

procedure-and-function-declaration-part ::=
                                    { procedure-or-function-declaration }

procedure-declaration :: = procedure-heading block ;

procedure-heading ::= procedure identifier ; | procedure identifier
                      ( formal-parameter-section
                      {, formal-parameter-section } ) ;

procedure-or-function-declaration ::= procedure-declaration |
                                      function-declaration

procedure-statement ::= procedure-identifier | procedure-identifier
                        ( actual-parameter {, actual-parameter } )

program-heading ::= empty | program program-identifier file-identifier-
list

program ::= program-heading block .

program-identifier ::=  identifier
```

```
record-constant ::= ( record-constant-element {; record-constant-element
} )

record-constant-element ::= field-identifier : structured-constant

record-section ::= empty | field-identifier {, field-identifier } : type

record-type ::= record field-list end

record-variable ::= variable

record-variable-list ::= record-variable {, record-variable }

referenced-variable ::= pointer-variable ^

relational-operator ::= = | <> | <= | >= | < | > | in

repeat-statement ::= repeat statement {; statement } until expression

repetitive-statement ::= while-statement | repeat-statement | for-
statement

result-type ::= type-identifier

scalar-type ::= ( identifier {, identifier } )

scale-factor ::= digit-sequence | sign digit-sequence

set ::= [ { set-element } ]

set-constant ::= [ { set-constant-element } ]

set-constant-element ::= constant | constant .. constant

set-element ::= expression | expression .. expression

set-type ::= set of base-type

sign ::= + | -

signed-factor ::= factor | sign factor

simple-expression ::= term { adding-operator term }

simple-statement ::= assignment-statement | procedure-statement |

                     goto-statement | inline-statement | empty-statement

simple-type ::= scalar-type | subrange-type | type-identifier

statement ::= simple-statement | structured-statement

statement-part ::= compound-statement

string ::= { string-element }

string-element ::= text-string | control-character
```

```
string-type ::= string [ constant ]

structured-constant ::= constant | array-constant | record-constant |

                        set-constant

structured-constant-definition ::= identifier : type = structured-
constant

structured-statement ::= compound-statement | conditional-statement |

                         repetitive-statement | with-statement

structured-type ::= unpacked-structured-type |

                    packed unpacked-structured-type

subrange-type ::= constant .. constant

tag-field ::= empty | field-identifier :

term ::= complemented-factor { multiplying-operator complemented-factor }

text-string ::= ' { character } '

type-definition ::= identifier = type

type-definition-part ::= type type-definition { ; type-definition } ;

type-identifier ::= identifier

type ::= simple-type | structured-type | pointer-type

typed-constant-identifier ::= identifier

unpacked-structured-type ::= string-type | array-type | record-type |

                             set-type | file-type

unsigned-constant ::= unsigned-number | string | constant-identifier |
nil

unsigned-integer ::= digit-sequence | $ hexdigit-sequence

unsigned-number ::= unsigned-integer | unsigned-real

unsigned-real ::= digit-sequence . digit-sequence |

                  digit-sequence . digit-sequence E scale-factor |

                  digit-sequence E scale-factor

untyped-constant-definition ::= identifier = constant

variable ::= entire-variable | compound-variable | referenced-variable

variable-declaration ::= identifier-list : type |
```

identifier-list : type absolute constant

variable-declaration-part ::= var variable-declaration

{; variable-declaration } ;

variable-identifier ::= identifier

variant ::= empty | case-label list : (field-list)

variant-part ::= case tag-field type-identifier of variant {; variant }

while-statement ::= while expression do statement

with-statement ::= with record-variable-list do statement

APPENDIX H

Overzicht VRAM adressering

Hieronder volgt een overzicht van de vier mogelijke coˆrdinaten stelsels die kunnen
worden gehanteerd voor een aantal van de GIOS procedures en functies die met
coˆrdinaten werken.

Pixel-georiÎnteerde adressering

Een eigenschap van een pixel-georiÎnteerde adressering is, dat men logische operaties
kan gebruiken, zie appendix I voor details over de te gebruiken logische operaties. Het
gebruik van logische operaties brengt altijd een verlies van performance met zich mee.

Een andere eigenschap is dat van alle kleur waarden die kunnen worden opgegeven (b.v.
via AtrByt) alleen de bits die nodig zijn voor ÈÈn pixel worden gebruikt. Voor scherm 5
zijn dit de bits 0-3, voor scherm 6 de bits 0-1, voor scherm 7 de bits 0-3 en voor scherm 8
de bits 0-7. Voor scherm 10 t/m 12 is dit geheel anders, maar de opgegeven kleur wordt
in ieder geval voor de volle 8 bits overgenomen.

Byte-georiÎnteerde adressering

Een eigenschap van een byte-georiÎnteerde adressering is, dat men geen logische
operaties kan gebruiken. Deze adresseer methode is daarom ook veel sneller dan de pixel-
georiÎnteerde adressering.

De tweede eigenschap van een byte-georiÎnteerde adressering is dat men afhankelijk van
het huidige scherm, 1 pixel, 2 pixels of 4 pixels leest of beschrijft. In het geval van 1 pixel
per byte (scherm 8 en hoger) is er geen beperking in de X-coˆrdinaten en alleen op de
logische operaties na is er ook geen verschil met een pixel-georiÎnteerde adressering.

In het geval van 2 pixels staat het linker pixel (linker 4 bits) altijd op een even

X-coˆrdinaat. Bij 4 pixels in ÈÈn byte staat het linker pixel (linker 2 bits) altijd op een

X-coˆrdinaat die deelbaar is door 2.

Als er een procedure als FastBox of FastCopy wordt gebruikt moet men er voor
zorgen dat de coˆrdinatenparen (X1,Y1) en (X2,Y2) altijd zodanig worden opgegeven
dat de kleinste X-coˆrdinaat altijd het meest linkse pixel aangeeft en de grootste

X-coˆrdinaat altijd het meest rechtse pixel. B.v. in scherm 6: X1 = 0 en X2 = 3 of
andersom.

Externe VRAM

Alle procedures en functies die in het GIOS aanwezig zijn en die ook maar iets doen met
het VRAM kunnen het externe VRAM aanspreken.

Dit extra VRAM heeft een aantal beperkingen en ÈÈn daarvan is dat het nooit op het
scherm weergegeven kan worden, maar dat wil niet zeggen dat het niet gebruikt kan
worden. In de tabel hieronder staat aangegeven welke pagina's er bij zijn gekomen om dit
externe VRAM te kunnen benutten.

Scherm

Normale pagina's

Extern VRAM pagina's

0 t/m 4

0, 1, 2 en 3

4 en 5

5

0, 1, 2 en 3

4 en 5

6

0, 1,2 en 3

4 en 5

7

0, 1

2

8, 10 t/m 12

0, 1

2

APPENDIX I

Overzicht logische operaties

Hieronder een overzicht van de mogelijke logische operaties die kunnen worden
opgegeven via de variabele LogOpr.

LogOpr

Operatie

Werking

0

IMP

DC:=SC

1

AND

DC:= SC and DC

2

OR

DC:=SC or DC

3

EOR

DC:=SC xor DC

4

NOT

DC:=not SC

5

-

6

-

7

-

8

TIMP

if SC = 0 DC:=DC else DC:=SC

9

TAND

if SC = 0 DC:=DC else DC:= SC and DC

10

TOR

if SC = 0 DC:=DC else DC:=SC or DC

11

TEOR

if SC = 0 DC:=DC else DC:=SC xor DC

12

TNOT

if SC = 0 DC:=DC else DC:=not SC

13

-

14

-

15

-

De variabele SC staat voor de bronkleur en variabele DC staat voor de doelkleur.

APPENDIX J

GIOS groeperingen

Hieronder een overzicht van alle GIOS procedures en functies ingedeeld per groep.

Grafisch

```
function GetClipping: Boolean;

function Point(X,Y: Integer): Byte;

function ReadStatus(StatusRegister: Byte): Byte;

function ReadVDP(VDPRegister: Integer): Byte;

function Search(X: Integer; Y: Byte;

                Color, Condition: Byte): Integer;

function Vpeek(Address: Integer): Byte;


procedure Bload(Offset: Integer; Page: Byte;

                Name: Str63);

procedure BSave(Address1:  Integer; Page1: Byte;

                Address2: Integer; Page2: Byte;

                Name: Str63);
```

```
procedure ChangeColor(ColorNr: Byte;
                        Red, Green, Blue: Byte);
procedure ChangeScreen(ScreenNr: Byte);
procedure Circle(X, Y, Radius: Integer);
procedure DefinePicture(X1, Y1, X2, Y2,
                          OffsetX, OffsetY: Integer;
                          Page: Byte);
procedure DisplayPage(PageNr: Byte);
procedure Ellipse(X, Y, RadiusX, RadiusY: Integer);
procedure Expand(X,Y: Integer; Page: Byte;
                 Name: Str63);
procedure Expand(X,Y: Integer; Page: Byte;
                 Name: Str63; Pos: Longint);
procedure FastBox(X1, Y1, X2, Y2: Integer);
procedure FastCopy(X1, Y1, X2, Y2,
                   DX, DY: Integer; DestPage: Byte);
procedure FillBox(X1, Y1, X2, Y2: Integer);
procedure FillShape(X: Integer; Y, Color: Byte);
procedure FillSprite(Plane, Color: Byte);
procedure GCopy(X1, Y1, X2, Y2: Integer;
                DX, DY: Integer; DestPage: Byte);
procedure GetViewPort(var X1, Y1, X2, Y2: Integer);
procedure Line(X1, Y1, X2, Y2: Integer);
procedure LoadPicture(X: Integer; Y, Page: Byte;
                        Name: Str63);
procedure LoadVRAM(Address: Integer; Page: Byte;
                   Position,Size: Real; Name: Str63);
procedure MoveVRAM(SAddress: Integer; SPage: Byte;
                   DAddress: Integer; DPage: Byte;
```

```
                    Size: Integer);
procedure Paint(X, Y: Integer; Border: Integer);
procedure PFillShape(X, Y: Integer);
procedure PPaint(X, Y: Integer; Border: Integer);
procedure Pset(X: Integer; Y: Byte);
procedure PutSprite(X, Y, SpriteNumber: Byte;
                    PlaneNumber: Byte);
procedure SavePicure(X1, Y1, X2, Y2: Integer;
                     Page: Byte; Name: Str63);
procedure Screen(ScreenNr: Byte);
procedure ScreenOff;
procedure ScreenOn;
procedure SetClipping(Flag: Boolean);
procedure SetViewPort(X1, Y1, X2, Y2: Integer);
procedure SpriteAttributeAddress(Address: Longint);
procedure SpriteColor(PlaneNumber: Byte;
                      Address: Integer);
procedure SpritePattern(SpriteNumber: Byte;
                        Address: Integer);
procedure SpritePatternAddress(Address: Longint);
procedure SpriteSize(Size: Byte);
procedure SpritesOff;
procedure SpritesOn;
procedure VPoke(Address: Integer; Data: Byte);
procedure WaitVDP;
procedure WriteVDP(Register, Data: Byte);
```

Memory mapper

```
function GetChannel(Channel: Byte): Real;
```

```
function GetPageID(Channel: Byte): Integer;
function SetMem(Pages: Integer): Integer;

procedure ClearMem;
procedure MemAppendFile(Channel: Byte;Position:Real;
                        Size: Real; Name: Str63);
procedure MemCopy(Source, Destination: Byte;
                  Length: Longint);
procedure MemExpand(X, Y: Integer;
                    Page, Channel: Byte);
procedure MemLoadPicture(X: Integer; Y: Byte;
                         Page, Channel: Byte);
procedure MemReadFile(Channel: Byte; Position: Real;
                      Size: Real; Name: Str63);
procedure MemSavePicure(X1, Y1, X2, Y2: Integer;
                        Page, Channel: Byte);
procedure MemWriteFile(Channel: Byte; Position: Real;
                       Size: Real; Name: Str63);
procedure ReadMem(Channel: Byte; Address: Integer;
                  Size: Integer);
procedure SetChannel(Channel: Byte; Position: Real);
procedure WriteMem(Channel: Byte;
                   Address: Integer; Size: Integer);
```

Systeem / DOS

```
function FindFirst(Path: Str63;
                   var Info: InfoBlock;
                   var Attribute: Byte;
                   var Name: Str63): Boolean;
```

```
function FindNext(var Info: InfoBlock;
                  var Attribute: Byte;
                  var Name: Str63): Boolean;
function GetDrive: Integer;
function GetError: Byte;
function SetDate(Year: Integer;
                 Month, Day: Byte): Boolean;
function SetTime(Hour, Minute,Second: Byte): Boolean;
function SimulatedDisk(DriveNr: Integer): Boolean;
function TestDrive(DriveNumber: Byte): Boolean;

procedure Date(var Year: Integer;
               var Month, Day, WeekDay: Byte);
procedure DeleteFile(FileName: Str63);
procedure GetDosVersion(var Kernel, System: Integer);
procedure ReadSector(Drive: Byte; Sector: Integer;
                     Address: Integer; Sectors:Byte);
procedure Time(var Hour, Minute, Second: Byte);
procedure WriteSector(Drive: Byte; Sector: Integer;
                      Address: Integer;Sectors:Byte);
```

Overig

```
function GetFKey: Integer;
function GetPad(Number: Integer): Integer;
function GetPdl(Num: Integer): Integer;
function ReadPSG(Register: Integer): Byte;
function Stick(Number: Byte): Byte;
function Strig(Number: Byte): Boolean;
function TsrPresent(Name: Str255): Boolean;
```

procedure Sound(Register, Value: Byte);

APPENDIX K

DOS 1 en DOS 2 Uitbreidingen

Deze appendix bevat alle gemaakte wijzigingen die zijn gemaakt voor het implementeren van een complete DOS 1 en DOS 2 ondersteuning.

Hieronder een korte lijst van de gemaakte wijzigingen:

- uitbreidingen in de syntax. Een aantal functies en procedures zijn nu uitgerust met longint parameters. De procedures Reset en ReWrite openen een file standaard met een blokgrootte van 128 bytes, maar deze kan optioneel worden opgegeven.

- complete herschrijving van alle file I/O procedures en functies voor gebruik onder DOS 1 en DOS 2.

- complete ondersteuning onder DOS 2 van padnamen.

- wegens de byte ge^rienteerde file I/O is het nu mogelijk om files te maken die veel groter kunnen zijn dan oorspronkelijk. De procedure Seek heeft nu als parameter een longint, waardoor men dus ruim boven de standaard 65535 records uit kan komen. Hierdoor zijn de getypeerde files niet meer compatibel. Een tweede voordeel hievan is dat de I/O zelf veel sneller is.

- uitgebreidere foutcontrole mogelijkheden. B.v. een 'disk not ready' fout kan nu worden afgevangen.

- alle fouten die in de oude versie werden doorgegeven d.m.v. IOResult zijn nu ook d.m.v. de GetError functie op te vragen. De GetError levert een gedetailleerdere foutmelding dan de IOResult doet. Ook de GIOS foutmeldingen zijn d.m.v. GetError uit te lezen.

- getypeerde files en ongetypeerde files hebben geen 128 bytes buffer meer. D.w.z. dat de procedure Flush geen nut meer heeft. Dit heeft als nadeel dat men bij een kleine blokgrootte een performance verslechtering heeft.

Padnaam

Hiermee wordt bedoeld een String van maximaal 63 tekens. Alles boven de 63 tekens wordt weggelaten.

Voorbeeld:

DOS 2 'A:\FRITS\OLAF\ERIK\test.dat'

DOS 1 'A:test.dat'

Let er op dat de backslash (\ ) wordt weggelaten onder DOS 1.

Gewijzigde procedures en functies

Assign

Syntax: Assign (FilVar, Str);

Str is een String expressie waarin een geldige file naam staat. Deze file naam wordt toegekend aan de file variabele FilVar. Alle verdere bewerkingen op FilVar zullen worden uitgevoerd naar de file Str. Assign mag nooit gebruikt worden op een reeds geopende file.

Onder DOS 2 mag men iedere gewenste padnaam invullen.

Hieronder een lijst met de mogelijke foutmeldingen die de Assign procedure kan geven.

IOResult	GetError	Omschrijving
$22	62	Assign to std files not allowed
$DB		Invalid drive
$DA		Invalid filename. Het eerste karakter van de filenaam heeft een ordinale waarde die kleiner is dan 32 of het karakter komt voor in de volgende reeks: '.,;:=/+"[]<>{}'

$D9

Invalid pathname. Geen hoofd filenaam aangegeven. B.v. 'A:\DIR1\.TXT'

Rewrite

Syntax: Rewrite (FilVar [, RecordSize]);

Een nieuwe lege file met de naam zoals die is toegekend aan FilVar wordt aangemaakt en
klaar gemaakt voor bewerking. De file wijzer wordt gezet op het begin van de file ofwel
op component 0. Een reeds bestaande file met dezelfde naam wordt overschreven.

RecordSize is 128 als deze niet wordt opgegeven. Het is zaak van de programmeur om er
voor te zorgen dat RecordSize een acceptabele inhoud heeft. RecordSize is alleen op te
geven als FilVar een ongetypeerde file is.

Reset

Syntax: Reset (FilVar [, RecordSize]);

De file met de naam zoals die is toegekend aan FilVar wordt klaar gemaakt voor
bewerking en de file wijzer wordt aan het begin van de file gezet (naar component 0).
FilVar moet de door Assign toegekende naam bevatten, anders treedt er een looptijdfout
op.

RecordSize is 128 als deze niet wordt opgegeven. Het is zaak van de programmeur om er
voor te zorgen dat RecordSize een acceptabele inhoud heeft. RecordSize is alleen op te
geven als FilVar een ongetypeerde file is.

De grote wijziging die hier gemaakt moest worden is dat de eerste 4 bytes die bij een
getypeerde file worden gebruikt om het aantal records en de record grootte te onthouden
nu overbodig zijn geworden. Het aantal records wordt nu uitgerekend door de file grootte
te delen door de record grootte van de getypeerde file. Verder moet gelden dat de
restwaarde van deze deling nul oplevert. Deze manier van werken is exact hetzelfde als
op de PC. Als dit niet overeenkomt dan wordt d.m.v. IOResult een 'Record length
mismatch' teruggegeven en GetError geeft dan de waarde 61 terug.

Read

Syntax: Read (FilVar, Varx);

Varx staat voor ÈÈn of meer variabelen van het component type van FilVar gescheiden door komma's. Elke variabele wordt gelezen uit de file en na ieder gelezen component wordt de file wijzer opgehoogd naar het volgende component.

Deze procedure is qua gebruik hetzelfde gebleven.

Write

Syntax: Write (FilVar, Varx);

Varx staat voor ÈÈn of meer variabelen van het component type van FilVar gescheiden door komma's. Elke variabele wordt geschreven naar de file en na ieder gelezen component wordt de file wijzer opgehoogd naar het volgende component.

Deze procedure is qua gebruik het zelfde gebleven.

Seek

Syntax: Seek (FilVar, N);

Seek verplaatst de file wijzer naar het component N van de file toegekend aan FilVar. N is een Longint expressie. De positie van het eerste component is 0. Merk op dat het mogelijk is om met Seek de file wijzer ÈÈn component voorbij het laatste component van de file te plaatsen. De opdracht:

```
  Seek (FilVar, FileSize (FilVar));
```

Plaatst de file wijzer aan het einde van de file (FileSize geeft het aantal componenten in de file weer, beginnend bij 0. Het terug gegeven aantal is dus ÈÈn groter dan het laatste component).

Flush

Syntax: Flush (FilVar);

Flush maakt de interne buffer van de file FilVar leeg en garandeerd zo dat de buffer is weggeschreven naar de file als er schrijfoperaties hebben plaats gevonden na de laatste file update. Flush zorgt er tevens voor dat de volgende leesoperatie fysiek uit de file wordt gelezen. Flush mag nooit gebruikt worden bij een gesloten file. Bij de opdracht Reset en Close wordt ook een Flush uitgevoerd.

Omdat alle lees- en schrijfopdrachten naar getypeerde en ongetypeerde files direct worden uitgevoerd, er vind dus geen buffering plaats, is de Flush procedure voor deze soorten overbodig.

Close

Syntax: Close (FilVar);

De file geassocieerd met FilVar wordt gesloten en bij een diskfile wordt de directory bij gewerkt. Merk op dat het nodig is om met Close een file te sluiten zelfs als er alleen uit de file is gelezen. Het aantal filehandles zal anders onder DOS 2.xx vlug opraken.

Deze procedure is qua gebruik het zelfde gebleven.

Erase

Syntax: Erase (FilVar);

De diskfile geassocieerd met FilVar wordt verwijderd. Als de diskfile nog open is, met andere woorden de diskfile is bewerkt met Reset of Rewrite maar nog niet gesloten, dan is het programmeertechnisch gezien beter de diskfile eerst te sluiten met Close voordat hij wordt verwijderd.

Als er een poging wordt gedaan om een device te wissen zal er via de functie GetError de waarde $C1 ('Invalid device operation') teruggegeven worden. Via IOResult wordt de overeenkomstige waarde $20 ('Operation not allowed on a logical device') teruggegeven.

Deze procedure is qua gebruik het zelfde gebleven.

Rename

Syntax: Rename (FilVar, Str);

De diskfile geassocieerd met FilVar krijgt een nieuwe naam welke is opgegeven met de String expressie Str. De disk directory wordt bijgewerkt en toont de nieuwe naam van de file. Alle verdere bewerkingen op FilVar worden uitgevoerd op de file met de nieuwe naam.

Waarschuwing: Rename mag nooit gebruikt worden bij een open file.

Opmerking: Het is de verantwoordelijkheid van de programmeur om ervoor te zorgen
dat de filenaam Str nog niet bestaat op de schijf. Als de naam wel bestaat kan dit leiden
tot looptijdfout.

Eof

Syntax: EOF (FilVar);

Dit is een booleaanse functie die True teruggeeft als de file wijzer naar de laatste
component van de file wijst. Zoniet dan geeft EOF de waarde False terug.

Deze procedure is qua gebruik het zelfde gebleven.

FilePos

Syntax: FilePos (FilVar);

Een functie die als waarde de positie van de file wijzer teruggeeft. Dit is altijd een getal
van het type Longint. Het eerste component van de file is 0.

Deze procedure is qua gebruik het zelfde gebleven.

FileSize

Syntax: FileSize (FilVar);

FileSize is een functie die een Longint getal teruggeeft dat het aantal componenten in de
file weergeeft. Als FileSize de waarde 0 opleverd is de file leeg.

Deze procedure is qua gebruik het zelfde gebleven.

Execute

Syntax: Execute (FilVar);

Deze procedure start de d.m.v. een Assign gekoppelde file vanaf het adres $0100. Ook
deze procedure is qua gebruik niet gewijzigd.

Chain

Syntax: Chain (FilVar);

Deze procedure start de d.m.v. een Assign gekoppelde 'chain' file. Ook deze procedure is
qua gebruik niet gewijzigd.

APPENDIX L

DOS 2 foutcodes

Hieronder volgt een complete lijst met DOS 2 foutcodes. Een aantal van deze codes

worden ook onder DOS 1 gegenereerd. B.v. 'Disk error', code 0FDH.

Incompatible disk (.NCOMP, 0FFh)

The disk cannot be accessed in that drive (eg. a double sided disk in a single sided drive).

Write error (.WRERR, 0FEh)

General error occurred during a disk write.

Disk error (.DISK, 0FDh)

General unknown disk error occurred.

Not ready (.NRDY, 0FCh)

Disk drive did not respond, usually means there is no disk in the drive.

Verify error (.VERFY, 0FBh)

With VERIFY enabled, a sector could not be read correctly after being written.

Data error (.DATA, 0FAh)

A disk sector could not be read because the CRC error checking was incorrect, usually
indicating a damaged disk.

Sector not found (.RNF, 0F9h)

The required sector could not be found on the disk, usually means a damaged disk.

Write protected disk (.WPROT, 0F8h)

Attempt to write to a disk with the write protect tab on.

Unformatted disk (.UFORM, 0F7h)

The disk has not been formatted, or it is a disk using a different recording technique.

Not a DOS disk (.NDOS, 0F6h)

The disk is formatted for another operating system and cannot be accessed by MSX-DOS.

Wrong disk (.WDISK, 0F5h)

The disk has been changed while MSX-DOS was accessing it. Must replace the correct disk.

Wrong disk for file (.WFILE, 0F4h)

The disk has been changed while there is an open file on it. Must replace the correct disk.

Seek error (.SEEK, 0F3h)

The required track of the disk could not be found.

Bad file allocation table (.IFAT, 0F2h)

The file allocation table on the disk has got corrupted. CHKDSK may be able to recover some of the data on the disk.

(.NOUPB, 0F1h)

This error has no message because it is always trapped internally in MSX-DOS as part of the disk change handling.

Cannot format this drive (.IFORM, 0F0h)

Attempt to format a drive which does not allow formatting. Usually as a result of trying
to format the RAM disk.

Internal error (.INTER, 0DFh)

Should never occur.

Not enough memory (.NORAM, 0DEh)

MSX-DOS has run out of memory in its 16k kernel data segment. Try reducing the
number of sector buffers or removing some environment strings. Also occurs if there
are no free segments for creating the RAMdisk.

Invalid MSX-DOS call (.IBDOS, 0DCh)

An MSX-DOS call was made with an illegal function number. Most illegal function
calls return no error, but this error may be returned if a "get previous error code"
function call is made.

Invalid drive (.IDRV, 0DBh)

A drive number parameter, or a drive letter in a drive/path/file string is one which does
not exist in the current system.

Invalid filename (.IFNM, 0DAh)

A filename string is illegal. This is only generated for pure filename strings, not drive/
path/file strings.

Invalid pathname (.IPATH, 0D9h)

Can be returned by any function call which is given an ASCIIZ drive/path/file
string. Indicates that the syntax of the string is incorrect in some way.

Pathname too long (.PLONG, 0D8h)

Can be returned by any function call which is given an ASCIIZ drive/path/file
string. Indicates that the complete path being specified (including current directory
if used) is longer than 63 characters.

File not found (.NOFIL, 0D7h)

Can be returned by any function which looks for files on a disk if it does not find one.
This error is also returned if a directory was specified but not found. In other cases,
.NODIR error (see below) will be returned.

Directory not found (.NODIR, 0D6h)

Returned if a directory item in a drive/path/file string could not be found.

Root directory full (.DRFUL, 0D5h)

Returned by "create" or "move" if a new entry is required in the root directory and it
is already full. The root directory cannot be extended.

Disk full (.DKFUL, 0D4h)

Usually results from a write operation if there was insufficient room on the disk for the
amount of data being written. May also result from trying to create or extend a sub-
directory if the disk is completely full.

Duplicate filename (.DUPF, 0D3h)

Results from "rename" or "move" if the destination filename already exists in the
destination directory.

Invalid directory move (.DIRE, 0D2h)

Results from an attempt to move a sub-directory into one of its own descendants. This

is not allowed as it would create an isolated loop in the directory structure.

Read only file (.FILRO, 0D1h)

Attempt to write to or delete a file which has the "read only" attribute bit set.

Directory not empty (.DIRNE, 0D0h)

Attempt to delete a sub-directory which is not empty.

Invalid attributes (.IATTR, 0CFh)

Can result from an attempt to change a file's attributes in an illegal way, or trying to do an operation on a file which is only possible on a sub-directory. Also results from illegal use of volume name fileinfo blocks.

Invalid . or .. operation (.DOT, 0CEh)

Attempt to do an illegal operation on the "." or ".." entries in a sub-directory, such as rename or move them.

System file exists (.SYSX, 0CDh)

Attempt to create a file or sub-directory of the same name as an existing system file. System files are not automatically deleted.

Directory exists (.DIRX, 0CCh)

Attempt to create a file or sub-directory of the same name as an existing sub-directory. Sub-directories are not automatically deleted.

File exists (.FILEX, 0CBh)

Attempt to create a sub-directory of the same name as an existing file. Files are not automatically deleted when creating sub-directories.

File already in use (.FOPEN, 0CAh)

Attempt to delete, rename, move, or change the attributes or date and time of a file
which has a file handle already open to it, other than by using the file handle itself.

Cannot transfer above 64K (.OV64K, 0C9h)

Disk transfer area would have extended above 0FFFFh.

File allocation error (.FILE, 0C8h)

The cluster chain for a file was corrupt. Use CHKDSK to recover as much of the file
as possible.

End of file (.EOF, 0C7h)

Attempt to read from a file when the file pointer is already at or beyond the end of file.

File access violation (.ACCV, 0C6h)

Attempt to read or write to a file handle which was opened with the appropriate access
bit set. Some of the standard file handles are opened in read only or write only mode.

Invalid process id (.IPROC, 0C5h)

Process id number passed to "join" function is invalid.

No spare file handles (.NHAND, 0C4h)

Attempt to open or create a file handle when all file handles are already in use. There
are 64 file handles available in the current version.

Invalid file handle (.IHAND, 0C3h)

The specified file handle is greater than the maximum allowed file handle number.

File handle not open (.NOPEN, 0C2h)

The specified file handle is not currently open.

Invalid device operation (.IDEV, 0C1h)

Attempt to use a device file handle or fileinfo block for an invalid operation such as
searching in it or moving it.

Invalid environment string (.IENV, 0C0h)

Environment item name string contains an invalid character.

Environment string too long (.ELONG, 0BFh)

Environment item name or value string is either longer than the maximum allowed
length of 255, or is too long for the user's buffer.

Invalid date (.IDATE, 0BEh)

Date parameters passed to "set date" are invalid.

Invalid time (.ITIME, 0BDh)

Time parameters passed to "set time" are invalid.

RAM disk (drive H:) already exists (.RAMDX, 0BCh)

Returned from the "ramdisk" function if trying to create a RAM disk when one already
exists.

RAM disk does not exist (.NRAMD, 0BBh)

Attempt to delete the RAM disk when it does not currently exist. A function which
tries to access a non-existent RAM disk will get a .IDRV error.

File handle has been deleted (.HDEAD, 0BAh)

The file associate with a file handle has been deleted so the file handle can no longer
be used.

(.EOL, 0B9h)

Internal error should never occur.

Invalid sub-function number (.ISBFN, 0B8h)

The sub-function number passed to the IOCTL function (function 4Bh) was invalid.

Ctrl-STOP pressed (.STOP, 09Fh)

The Ctrl-STOP key is tested in almost all places in the system including all character I/O.

Ctrl-C pressed (.CTRLC, 09Eh)

Ctrl-C is only tested for on those character functions which specify status checks.

Disk operation aborted (.ABORT, 09Dh)

This error occurs when any disk error is aborted by the user or automatically by
the system. The original disk error code will be passed to the abort routine in B as the
secondary error code.

Error on standard output (.OUTERR, 09Ch)

Returned if any error occurred on a standard output channel while it was being accessed
through the character functions (functions 01h...0Bh). The original error code is
passed to the abort routine in register B as the secondary error code. This error will
normally only occur if a program has altered the standard file handles.

Error on standard input (.INERR, 09Bh)

Returned if any error occurred on a standard input channel while it was being accessed through the character functions (functions 01h...0Bh). The original error code is passed to the abort routine in register B as the secondary error code. The most likely error is end of file (.EOF). This error will normally only occur if a program has altered the standard file handles.